

DR

DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota

$$\frac{\partial L}{\partial W_{\ell_1}}$$

$$\frac{\partial L}{\partial W_{\ell_2}}$$

$$\frac{\partial L}{\partial W_{\ell_3}}$$

$$\frac{\partial L}{\partial W_{\ell_4}}$$

$$\frac{\partial L}{\partial W_{\ell_5}}$$

$$\frac{\partial L}{\partial \text{Out}}$$



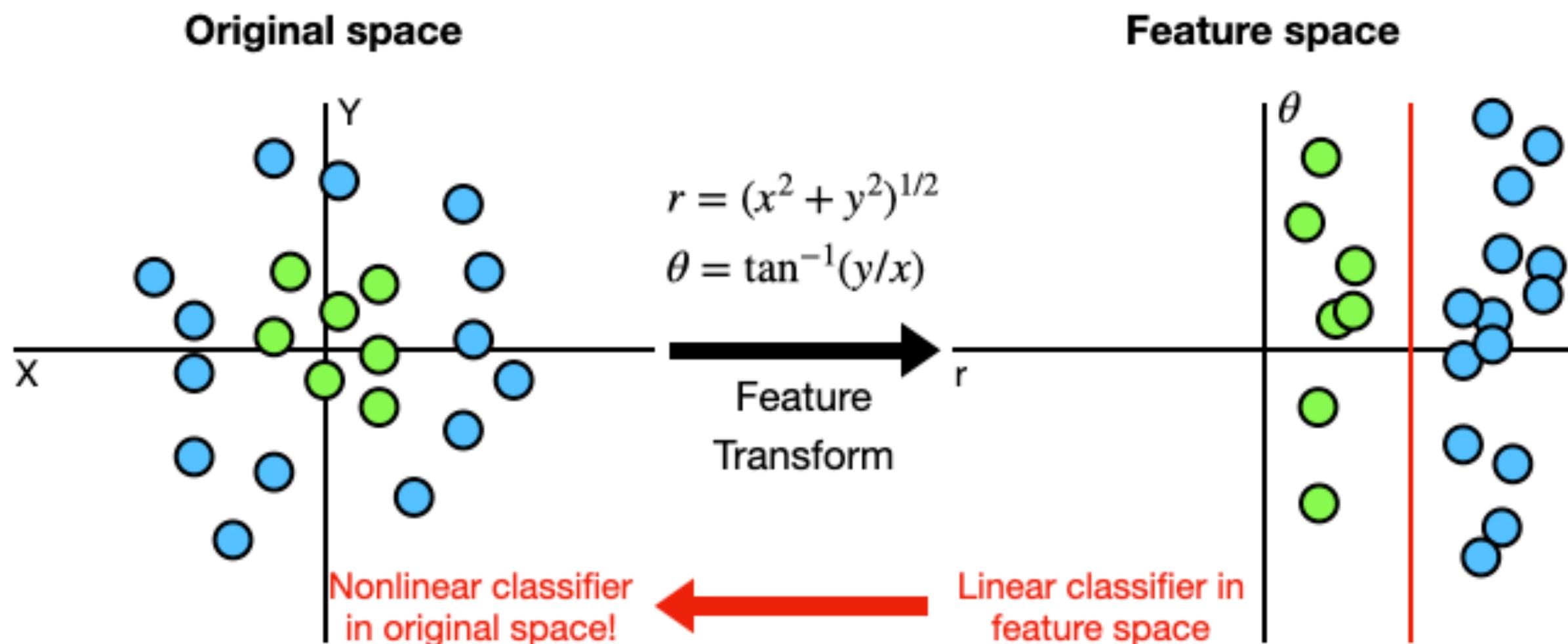
Project 1 – Reminder

- Instructions and code available on the website
 - Here: deeprob.org/projects/project1/
- Uses Python, PyTorch and Google Colab
- Implement KNN, linear SVM, and linear softmax classifiers
- **Autograder is online and updated**
- **Due Thursday, January 26th 11:59 PM EST**

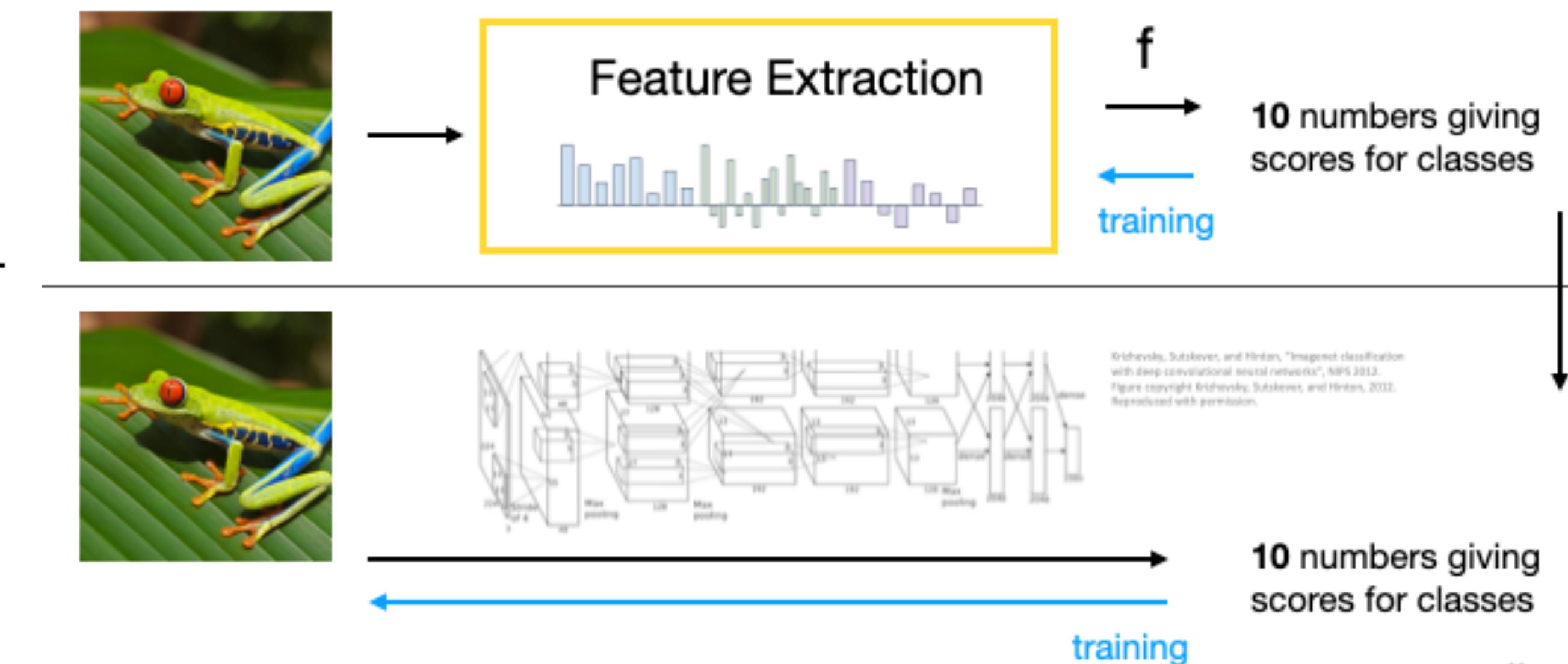


Recap from Previous Lecture

Feature transform + Linear classifier allows nonlinear decision boundaries



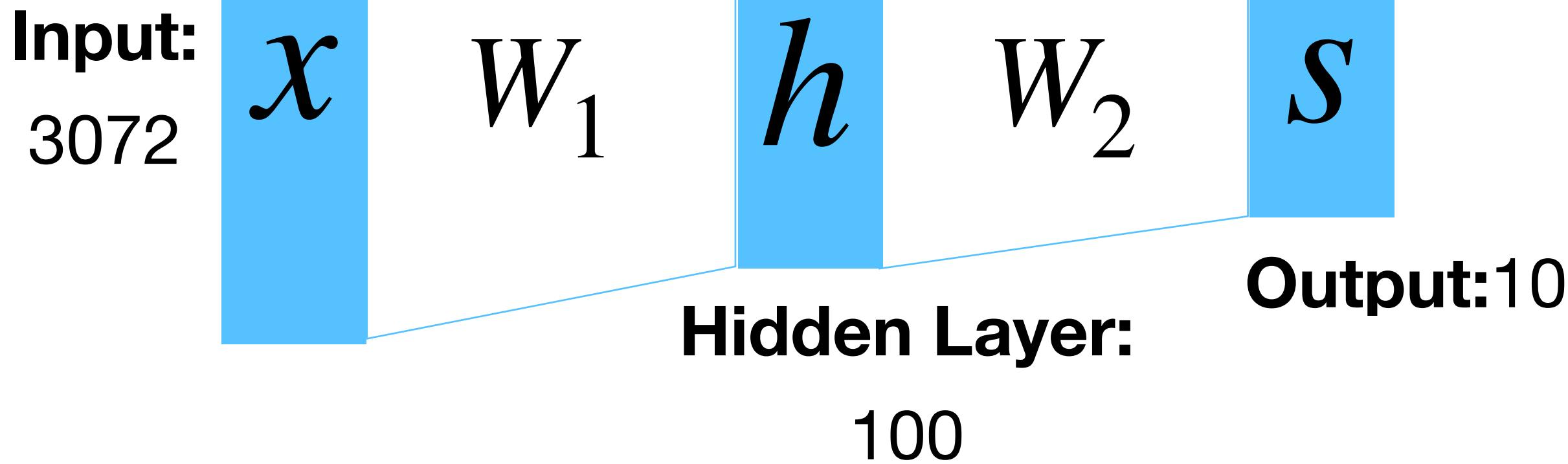
Neural Networks as learnable feature transforms



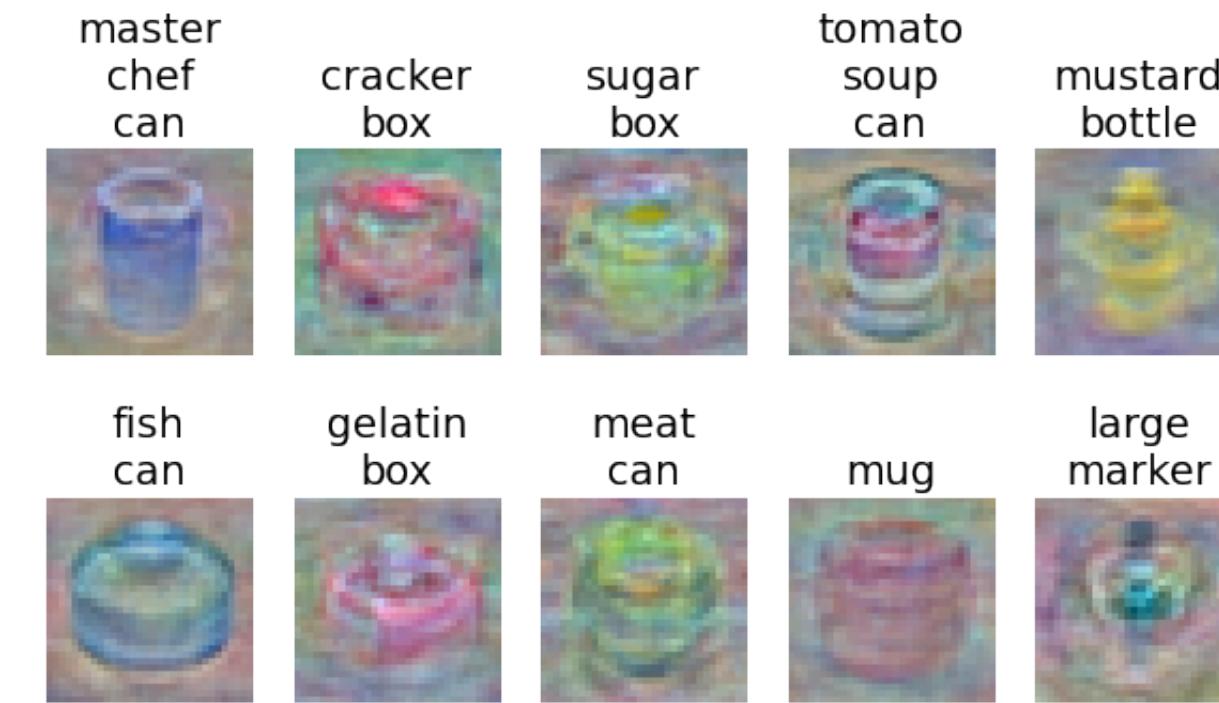
Recap from Previous Lecture

From linear classifiers to
fully-connected networks

$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



Linear classifier: One template per class



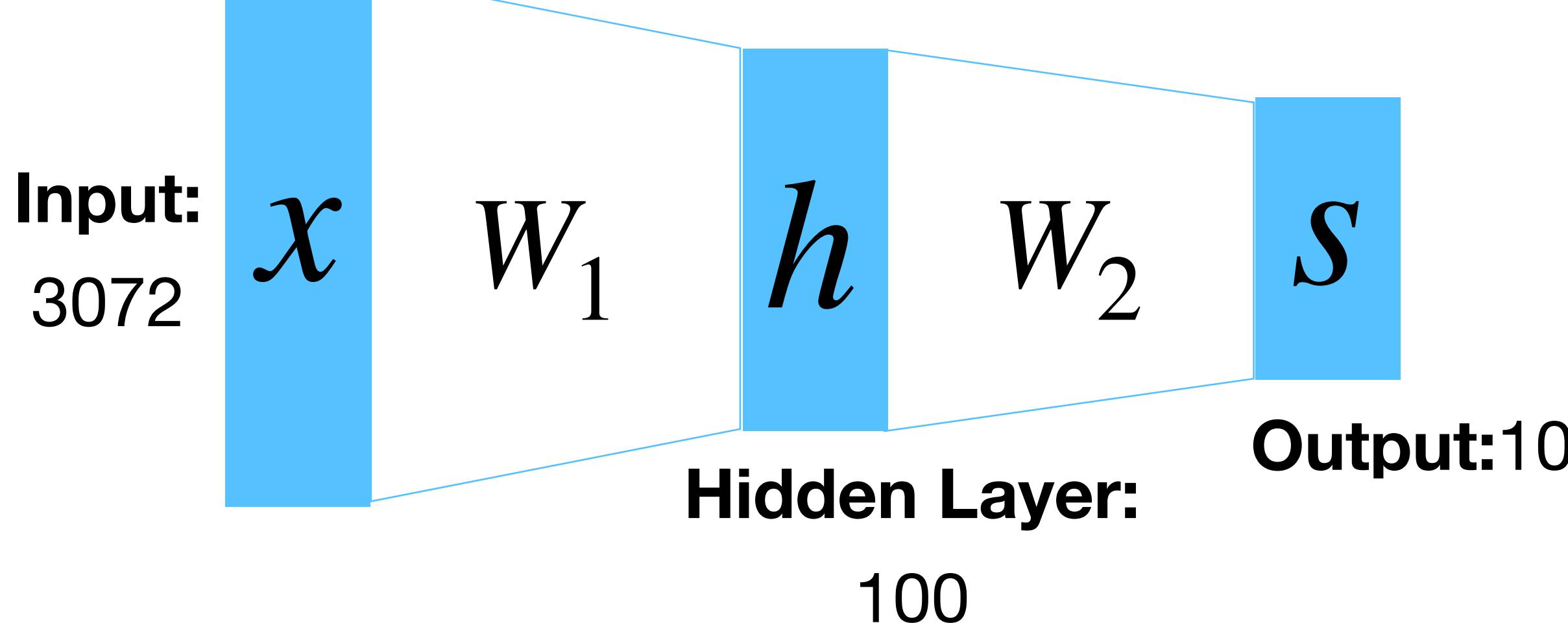
Neural networks: Many reusable templates



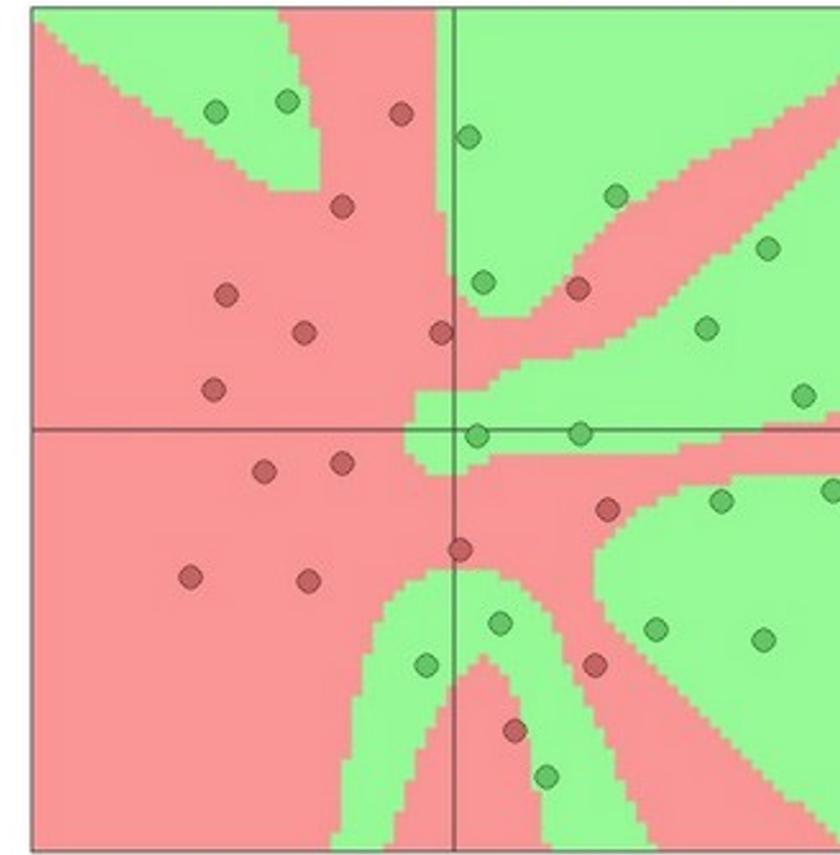
Recap from Previous Lecture

From linear classifiers to
fully-connected networks

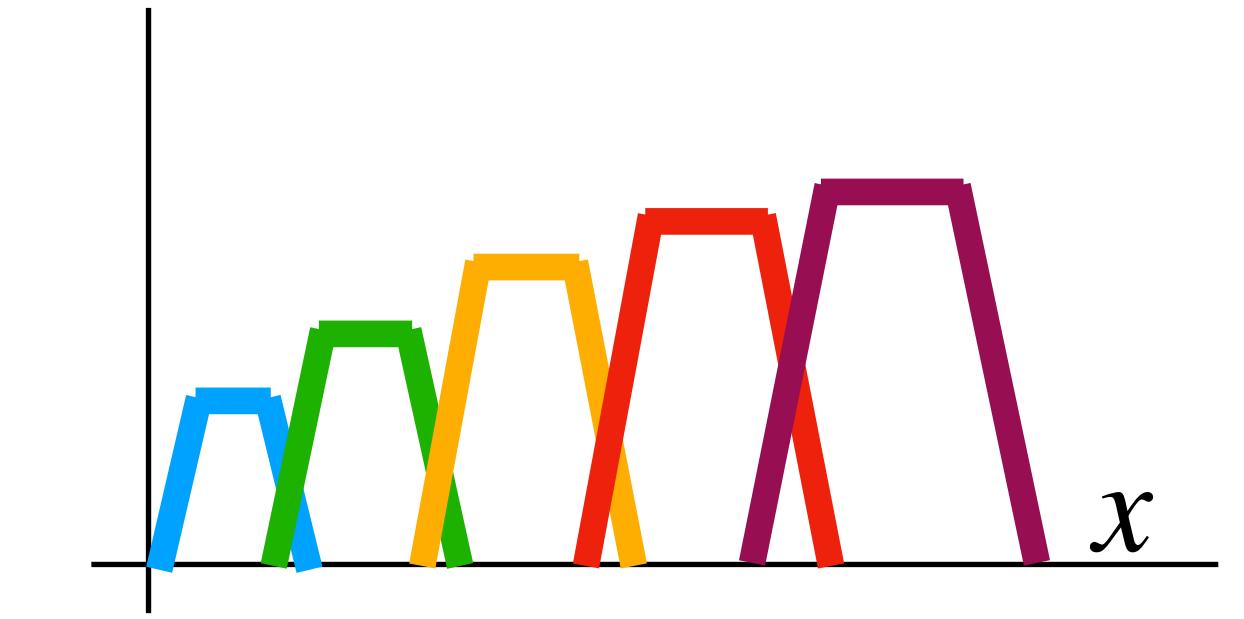
$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



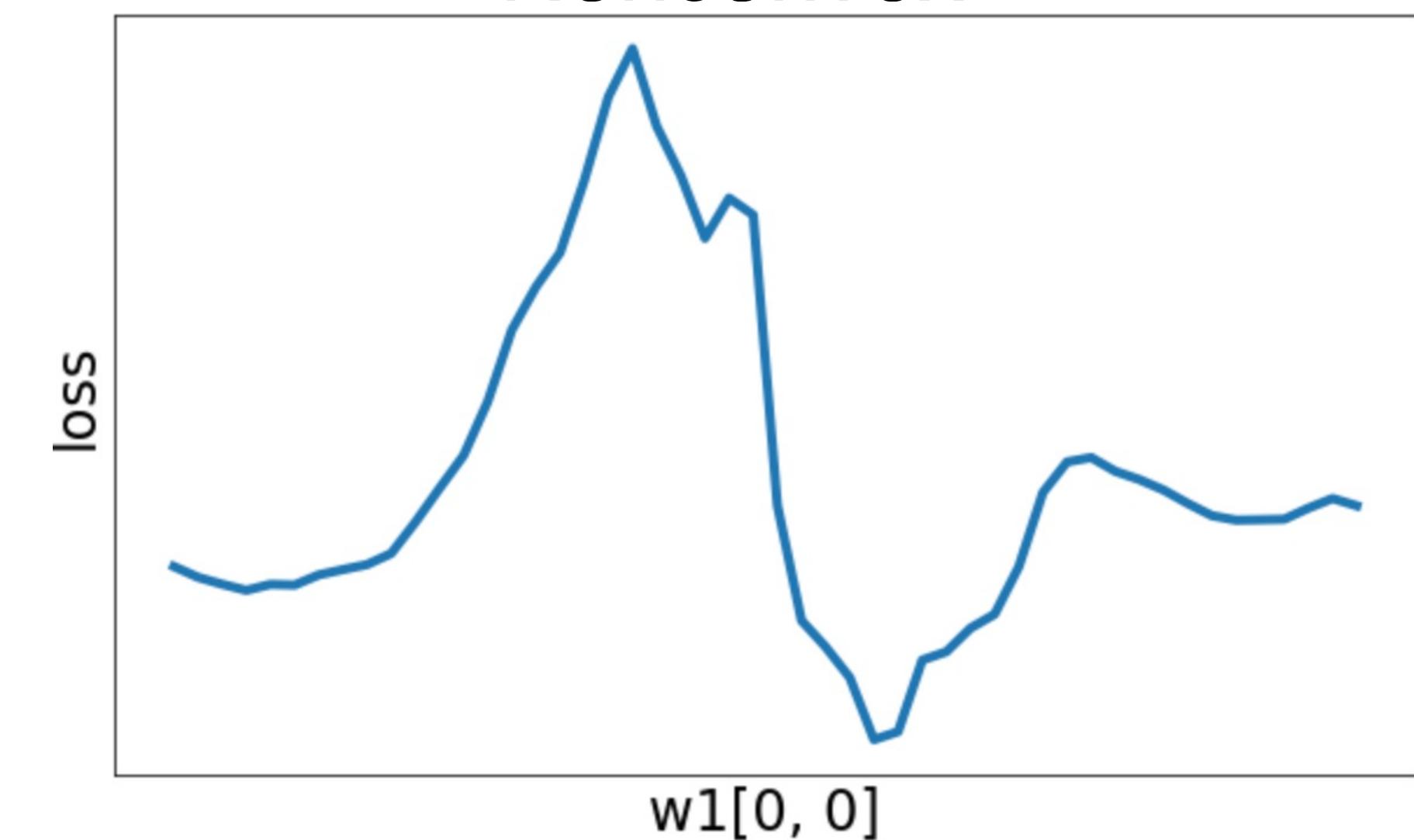
Space Warping



Universal approximation



Nonconvex



Problem: How to compute gradients?

$$s = W_2 \max(0, W_1 x + b_1) + b_2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$R(W) = \sum_k W_k^2$$

$$L(W_1, W_2, b_1, b_2) = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(W_1) + \lambda R(W_2)$$

If we can compute $\frac{\delta L}{\delta W_1}, \frac{\delta L}{\delta W_2}, \frac{\delta L}{\delta b_1}, \frac{\delta L}{\delta b_2}$ then we can optimize with SGD

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) \\ L &= \frac{1}{N} \sum_{i=1}^N L_i + \lambda \sum_k W_k^2 \\ &= \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) + \lambda \sum_k W_k^2 \end{aligned}$$

Problem: Very tedious with lots of matrix calculus

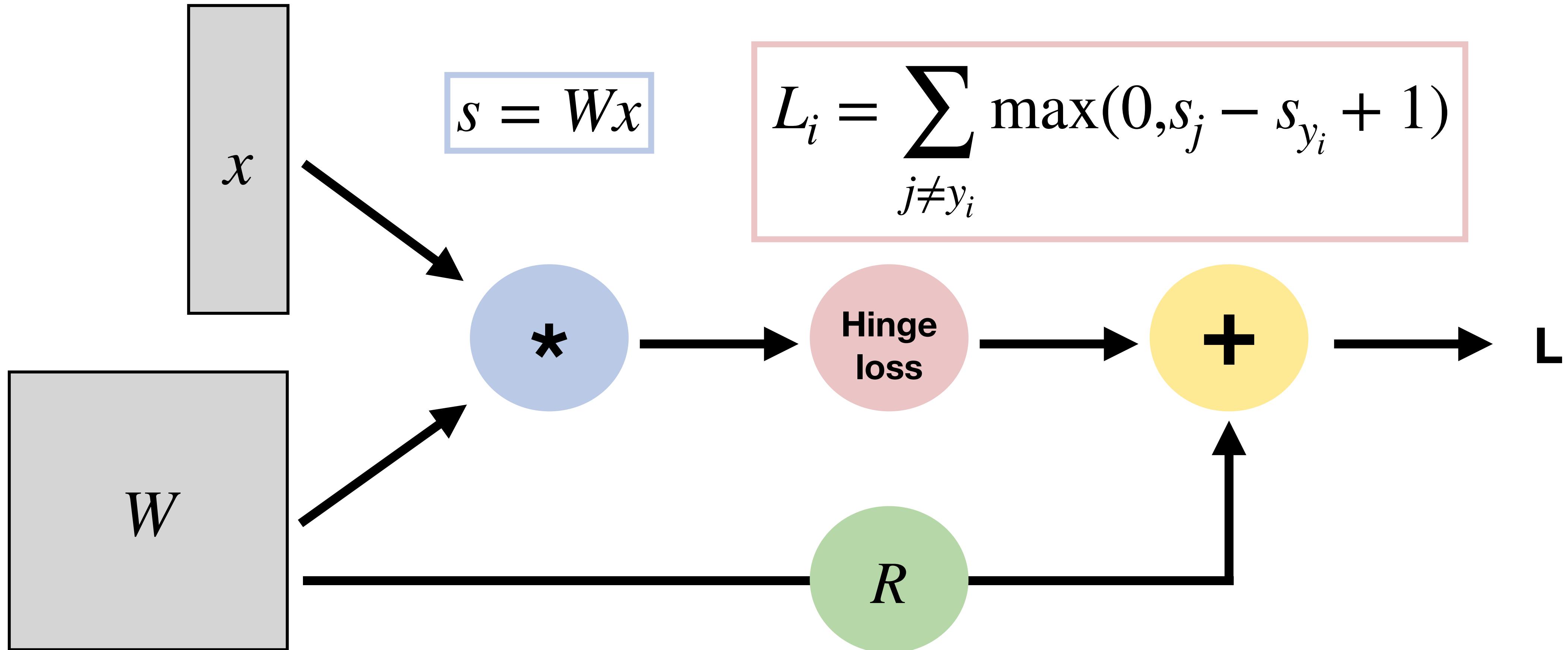
Problem: What if we want to change the loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

Problem: Not feasible for very complex models!

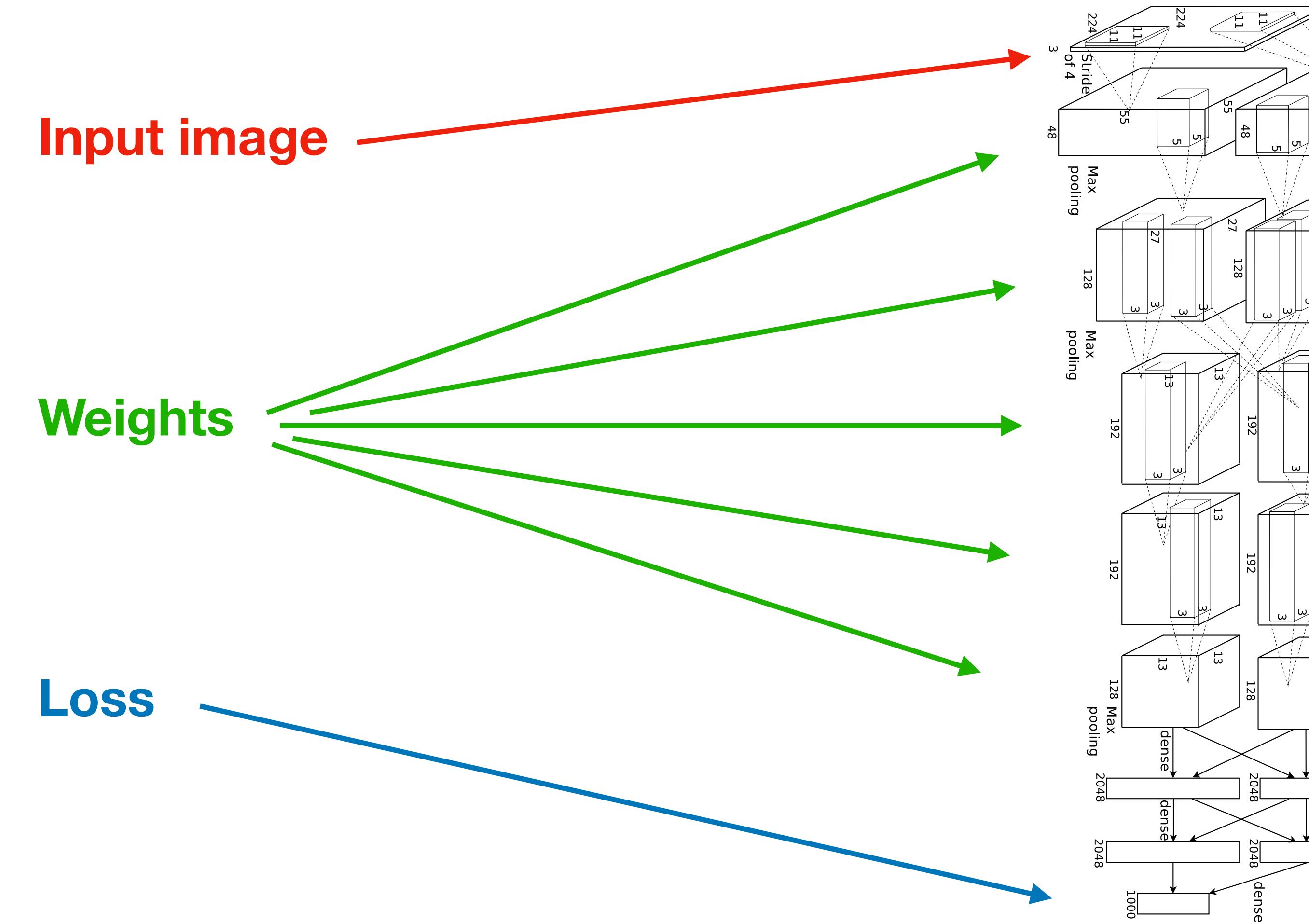
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, W_{j,:} x - W_{y_i,:} x + 1) + \lambda \sum_k W_k^2 \right)$$



Better Idea: Computational Graphs

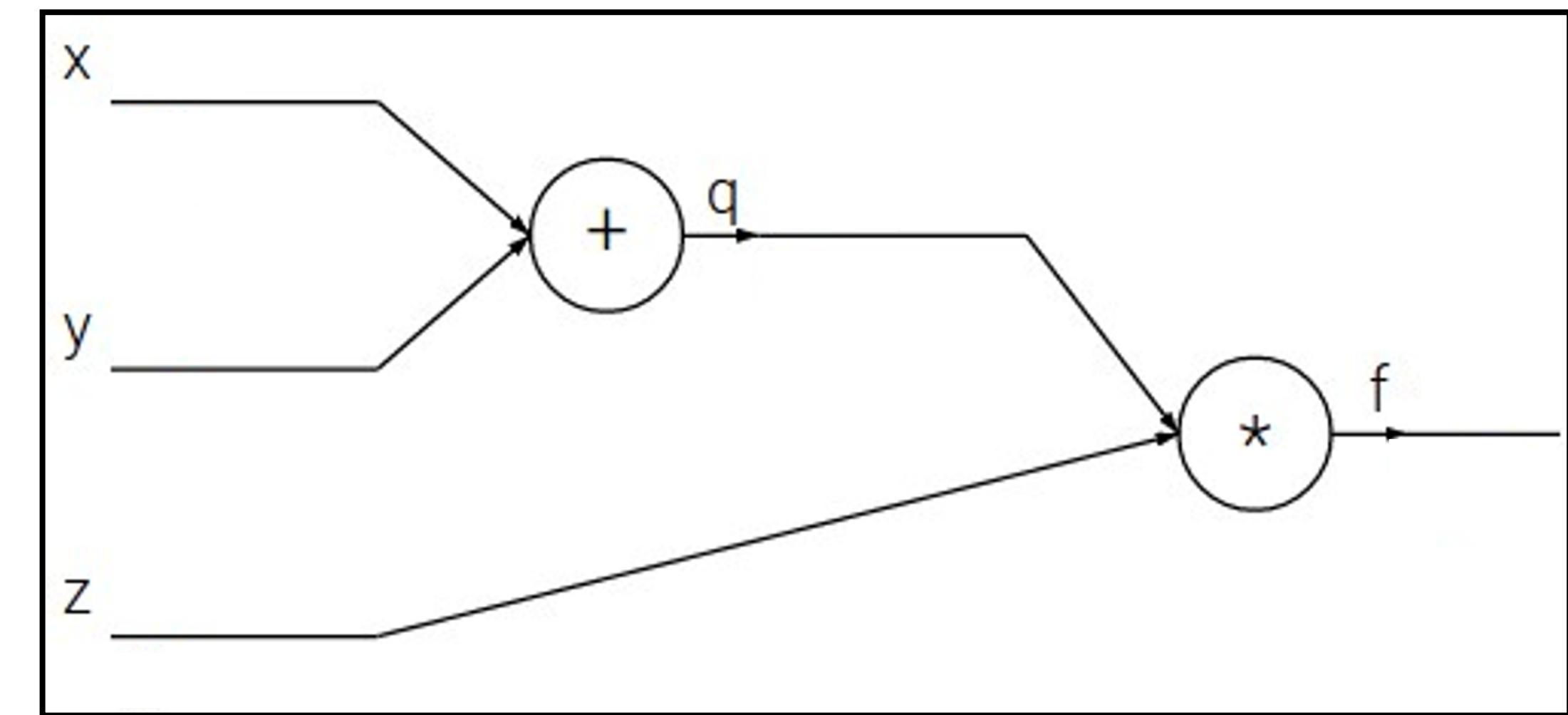


Deep Network (AlexNet)



Backpropagation: Simple Example

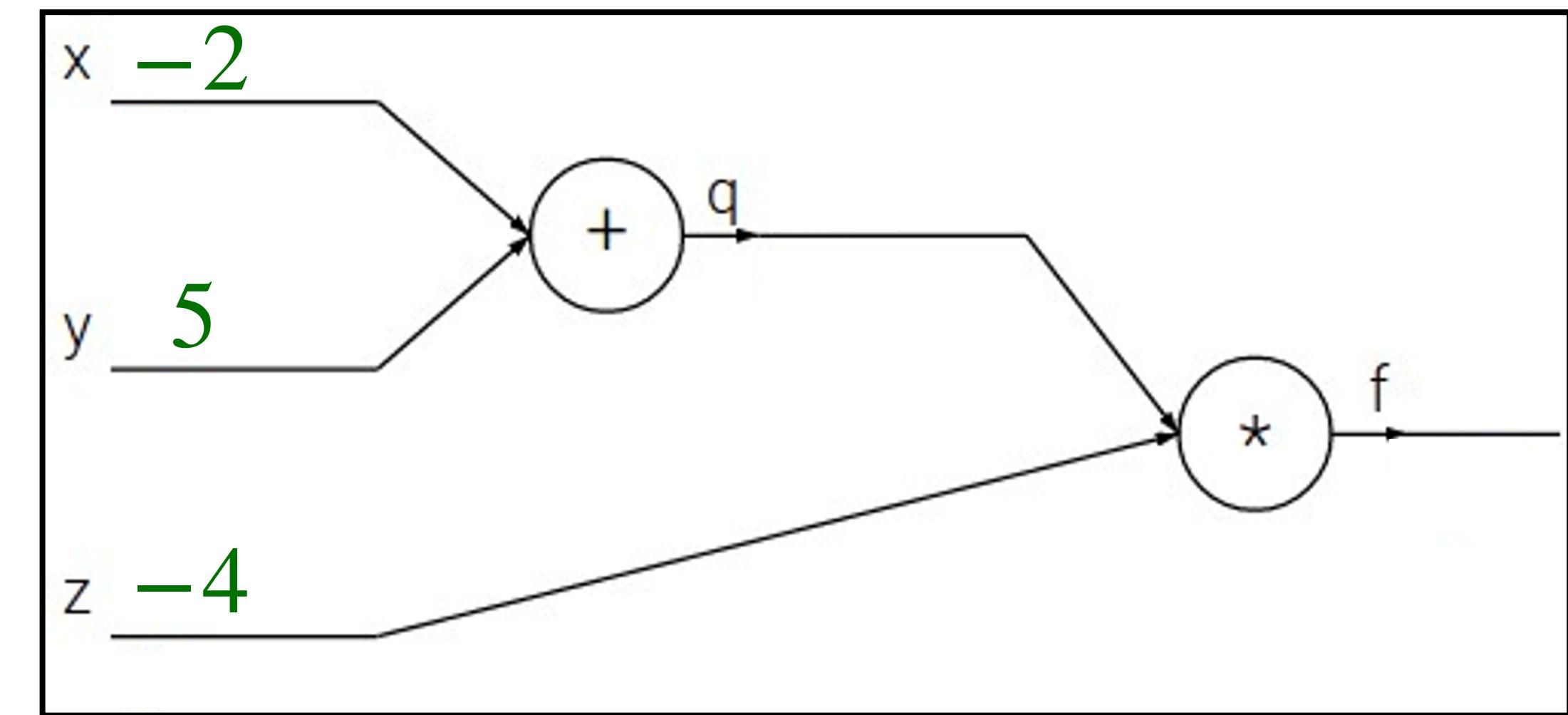
$$f(x, y, z) = (x + y) \cdot z$$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. $x = -2, y = 5, z = -4$



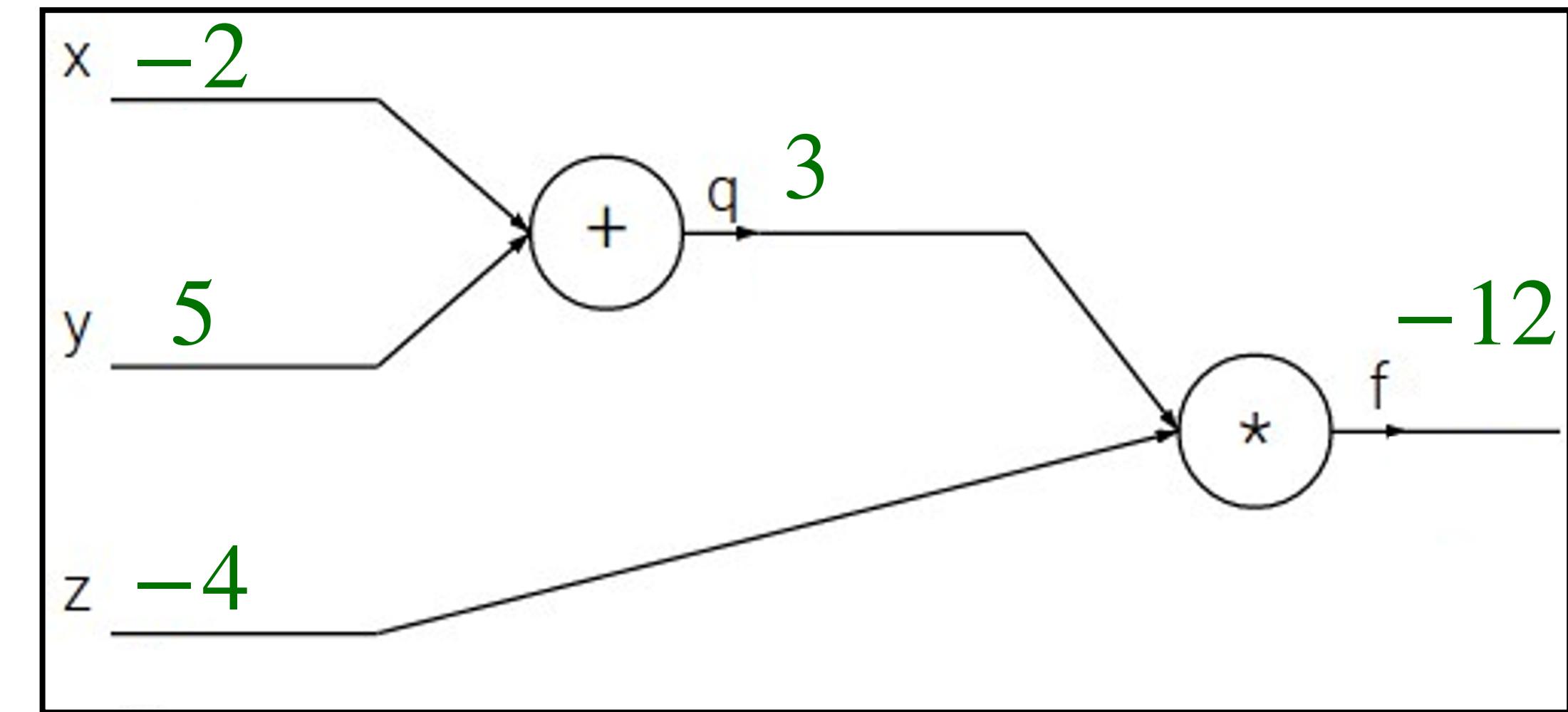
Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

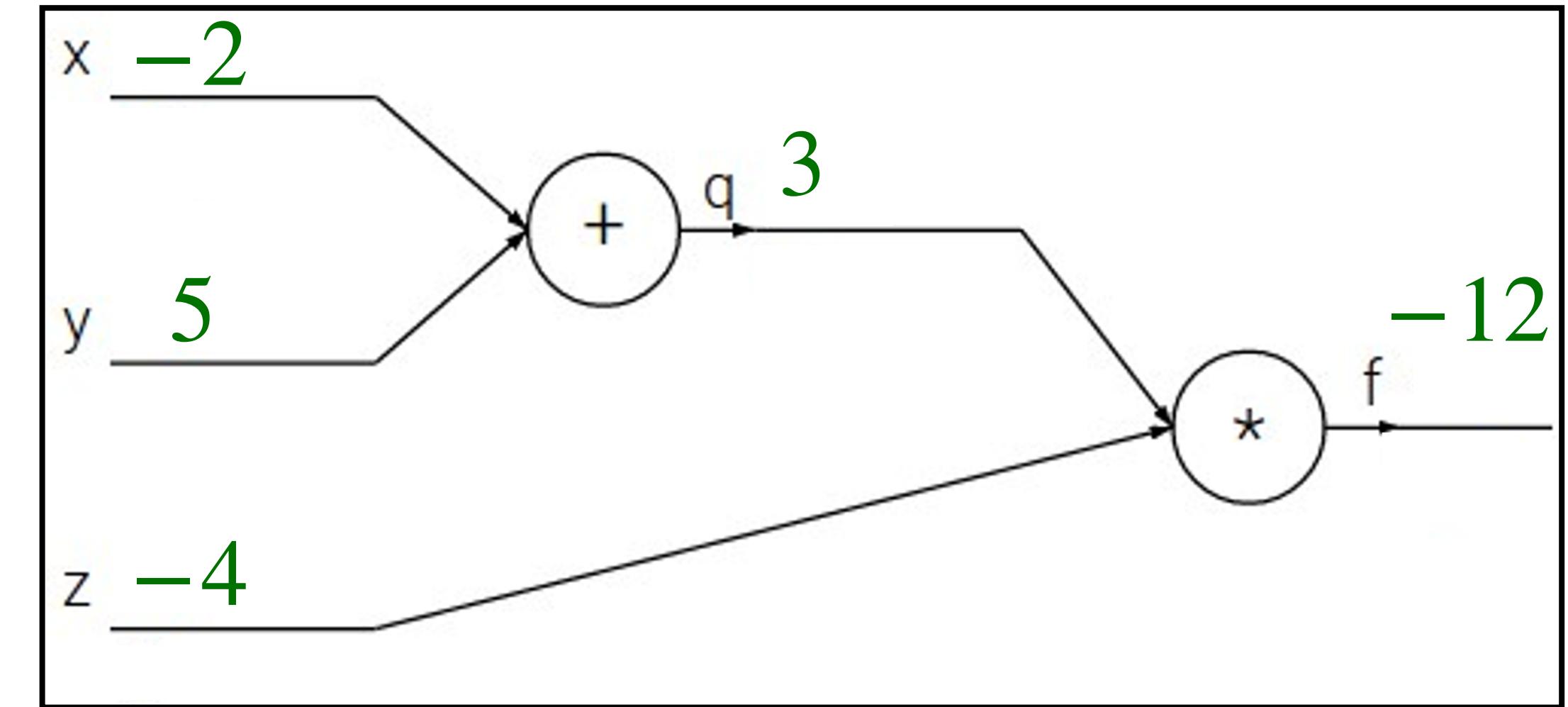
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

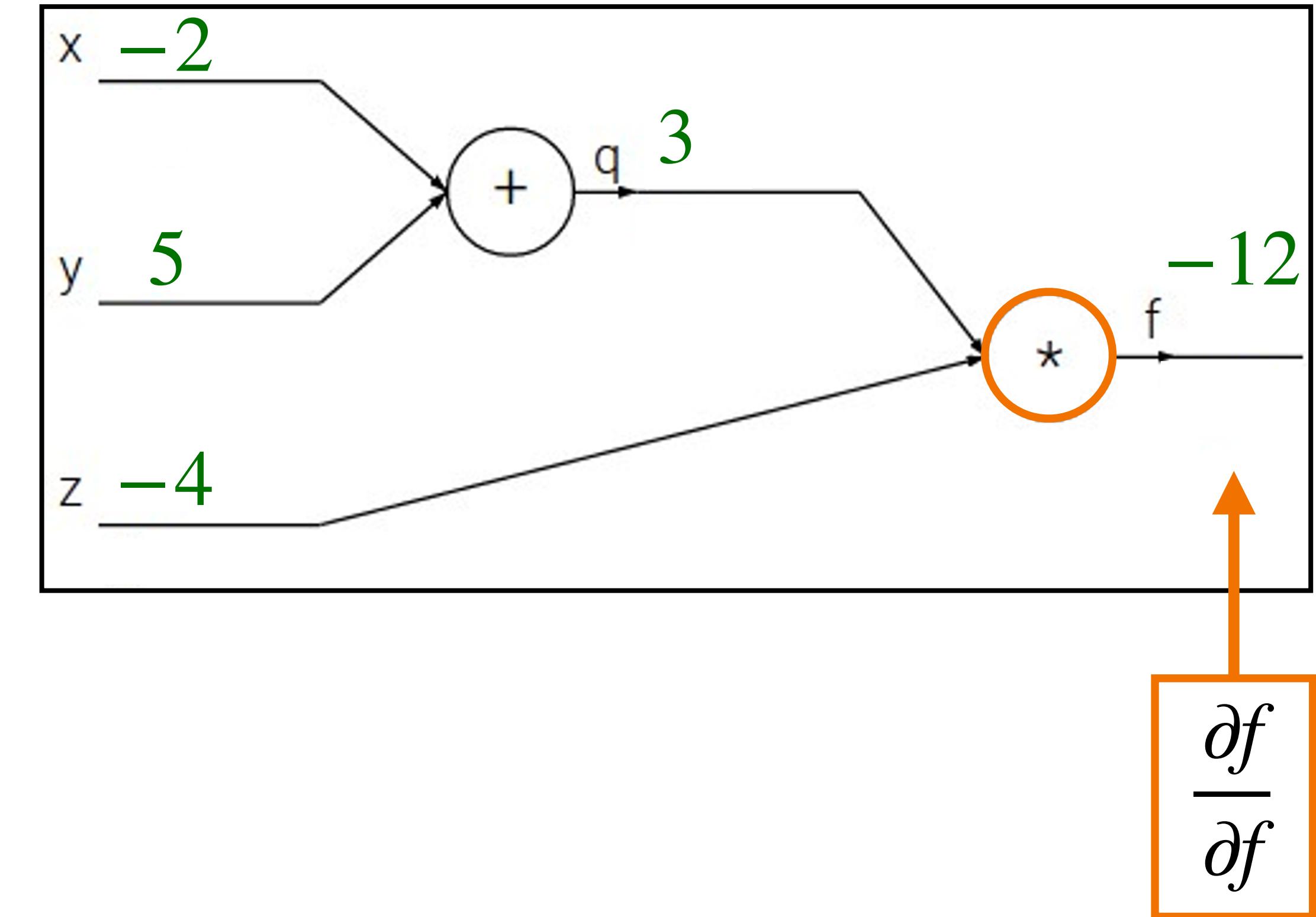
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

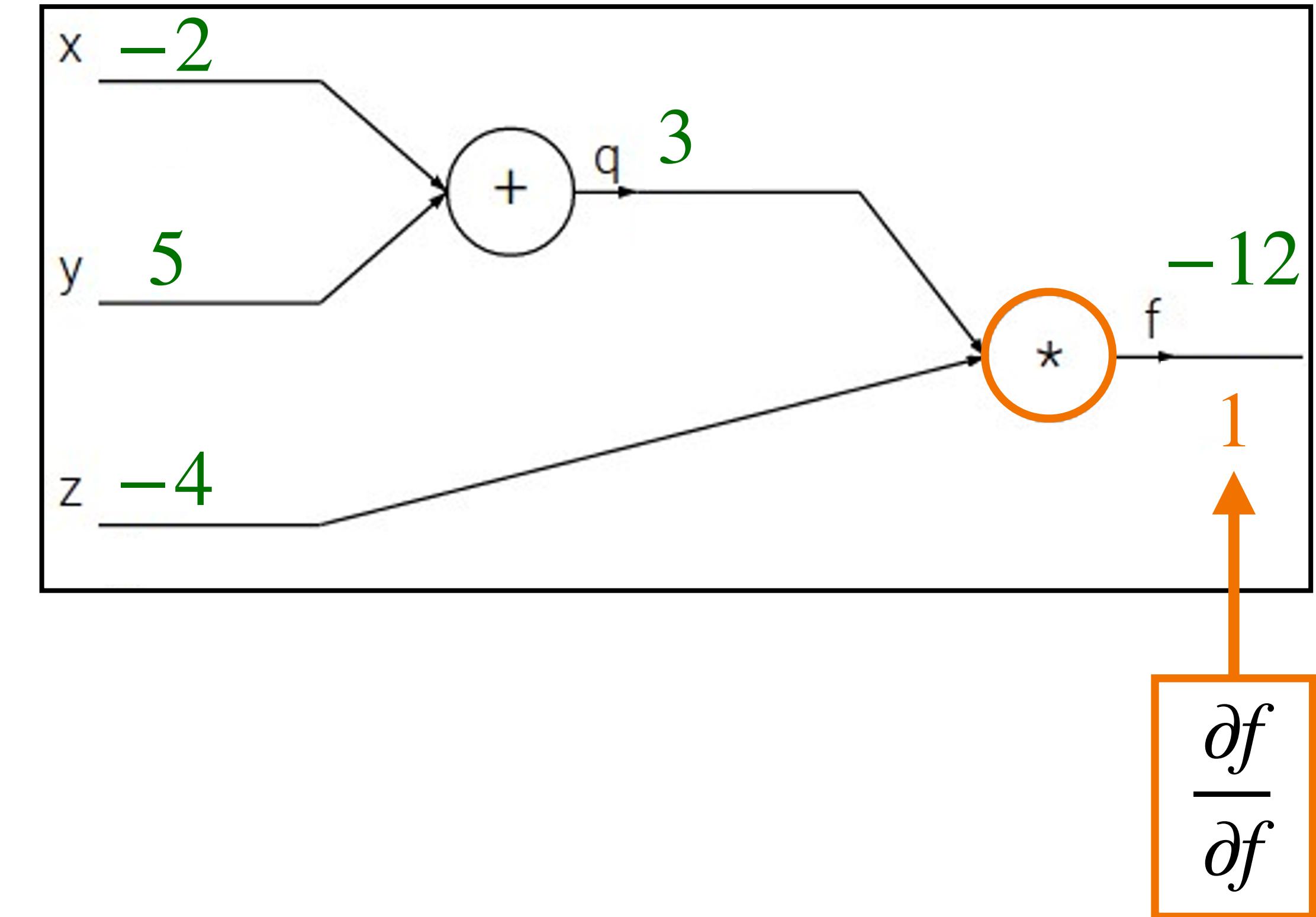
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

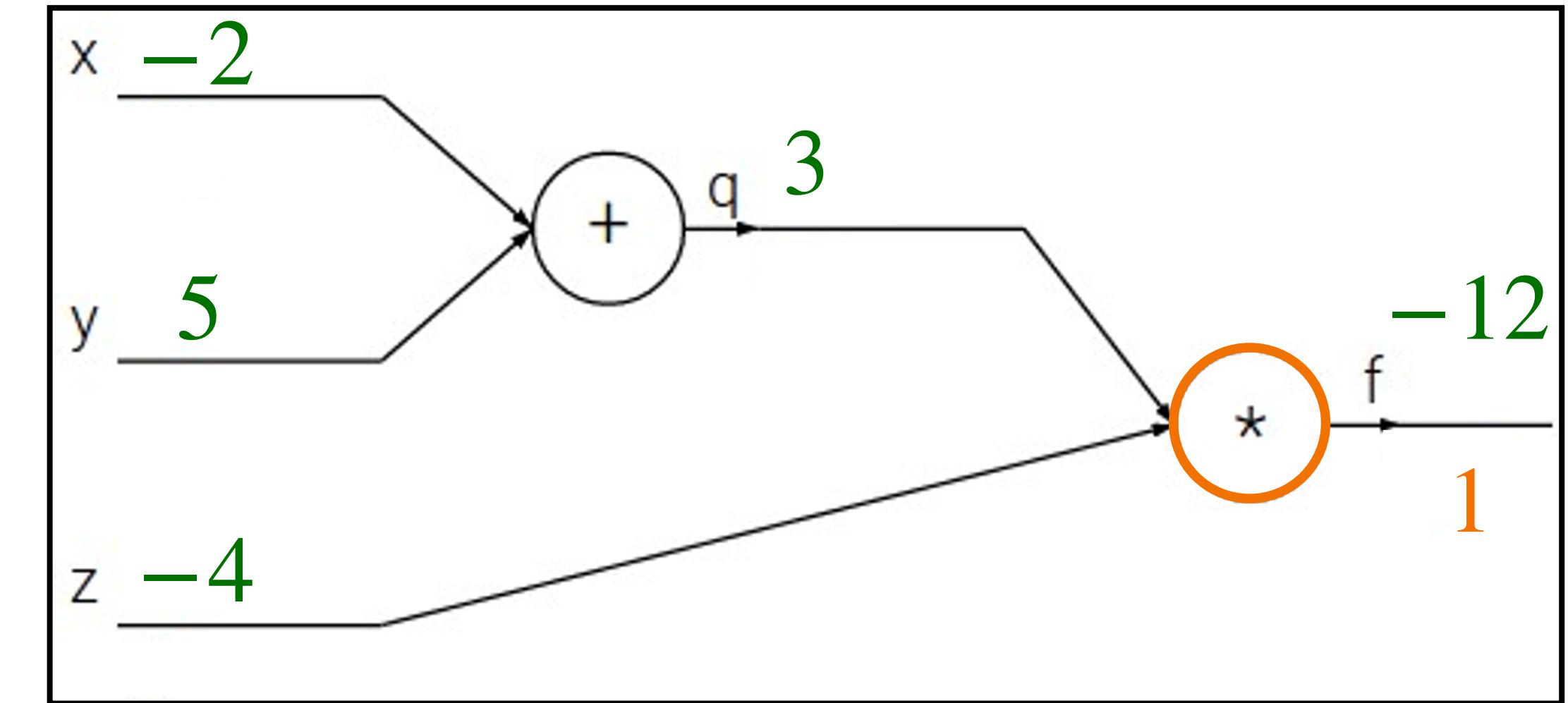
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

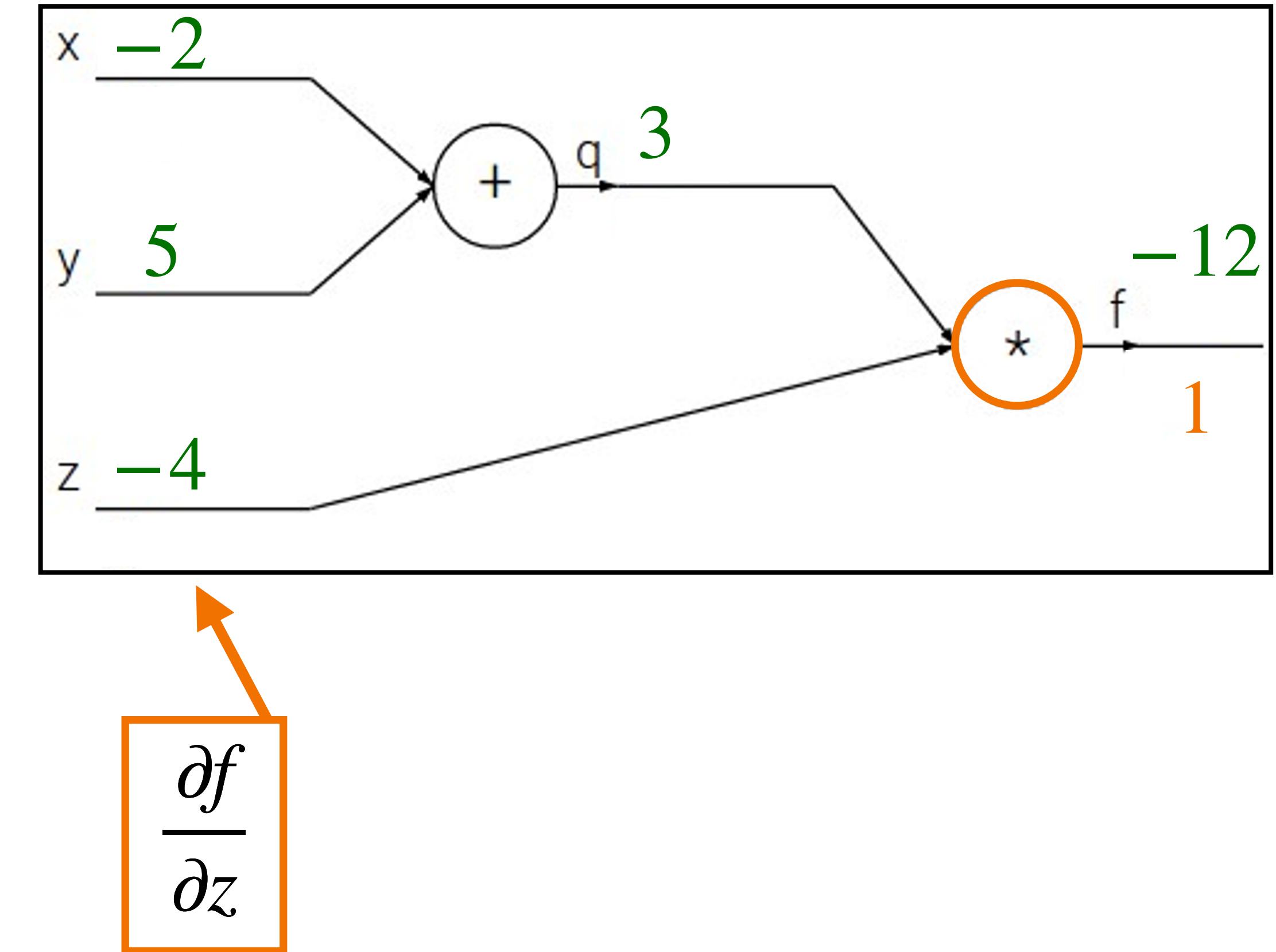
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

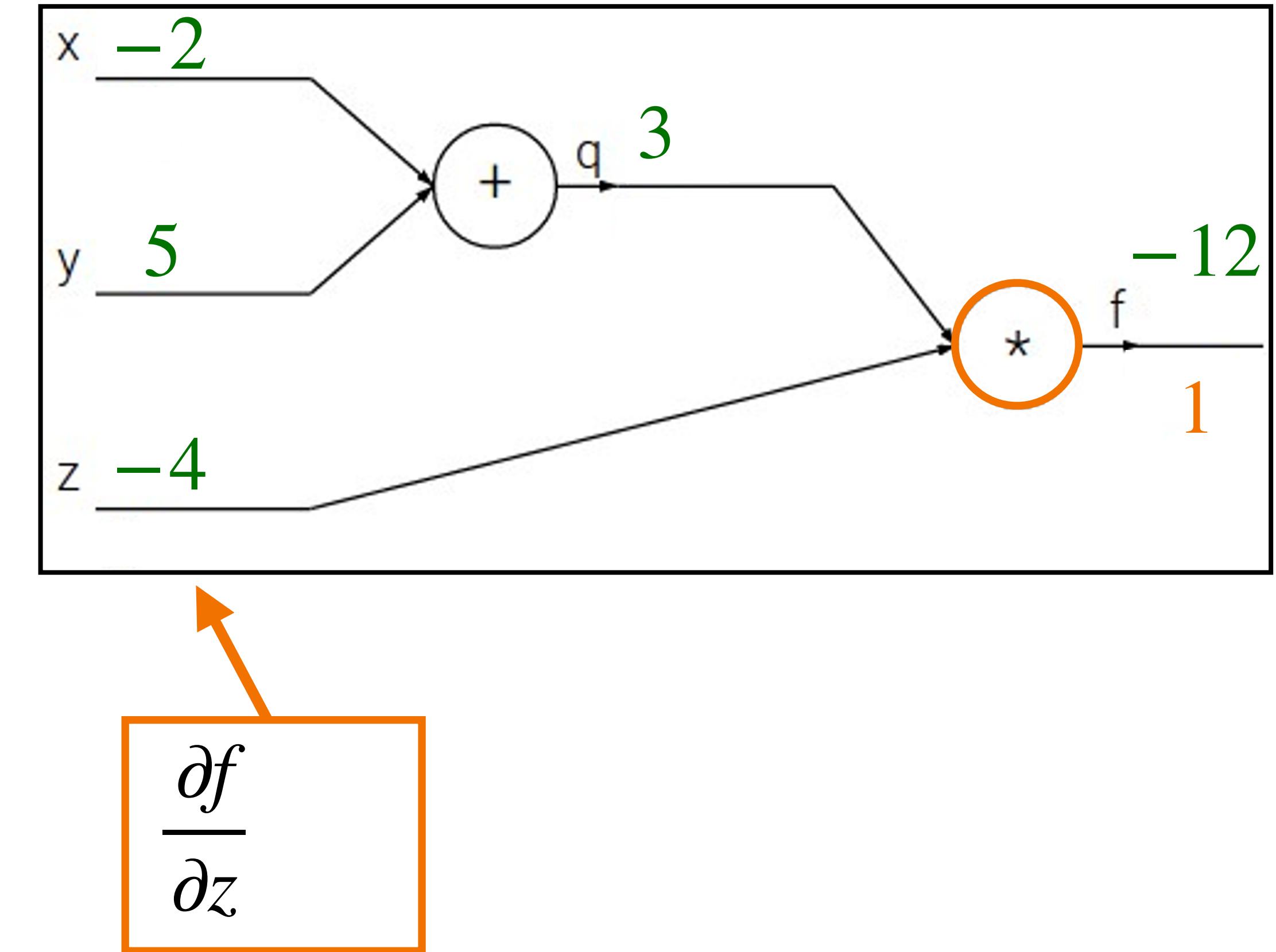
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

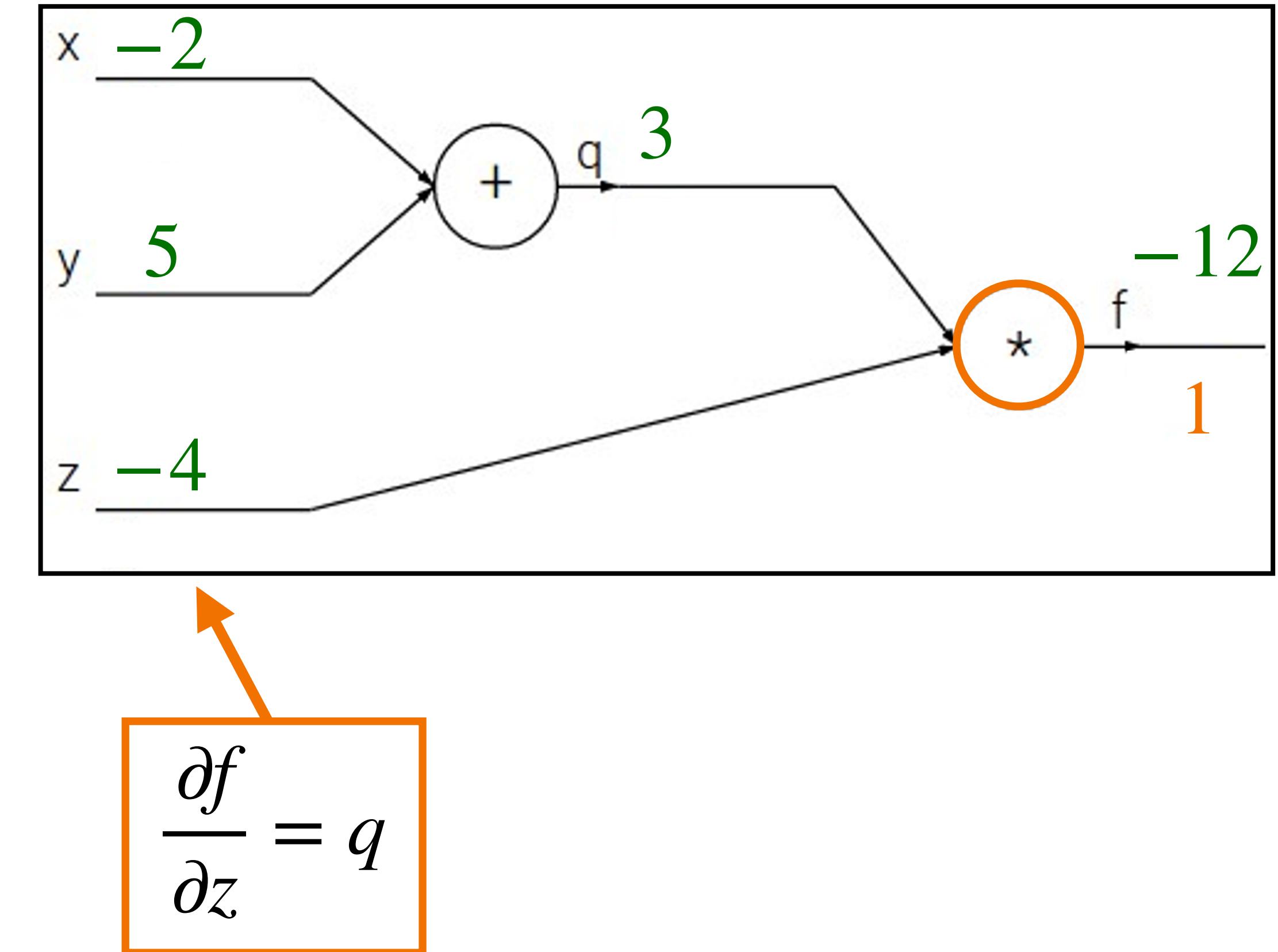
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

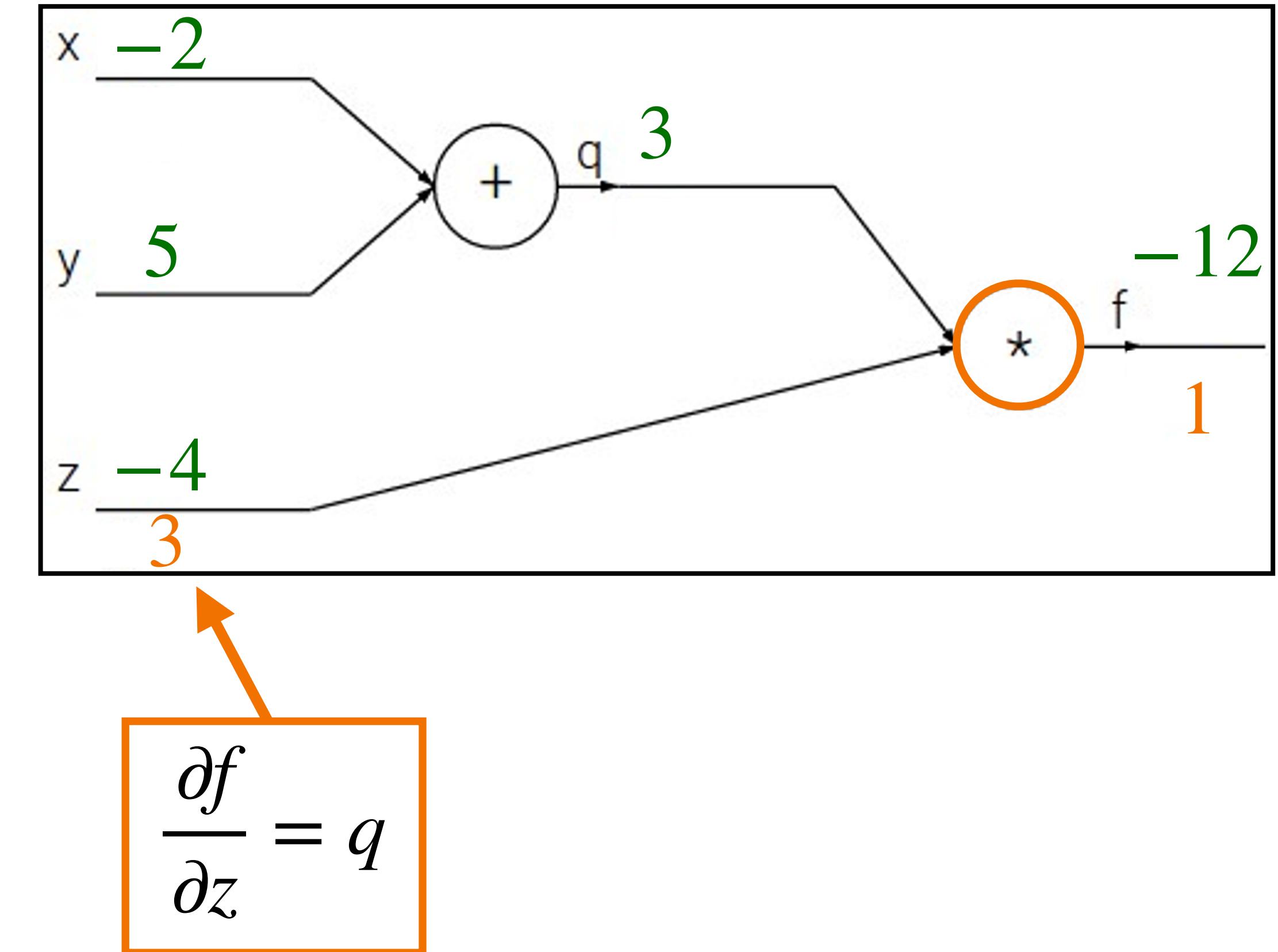
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

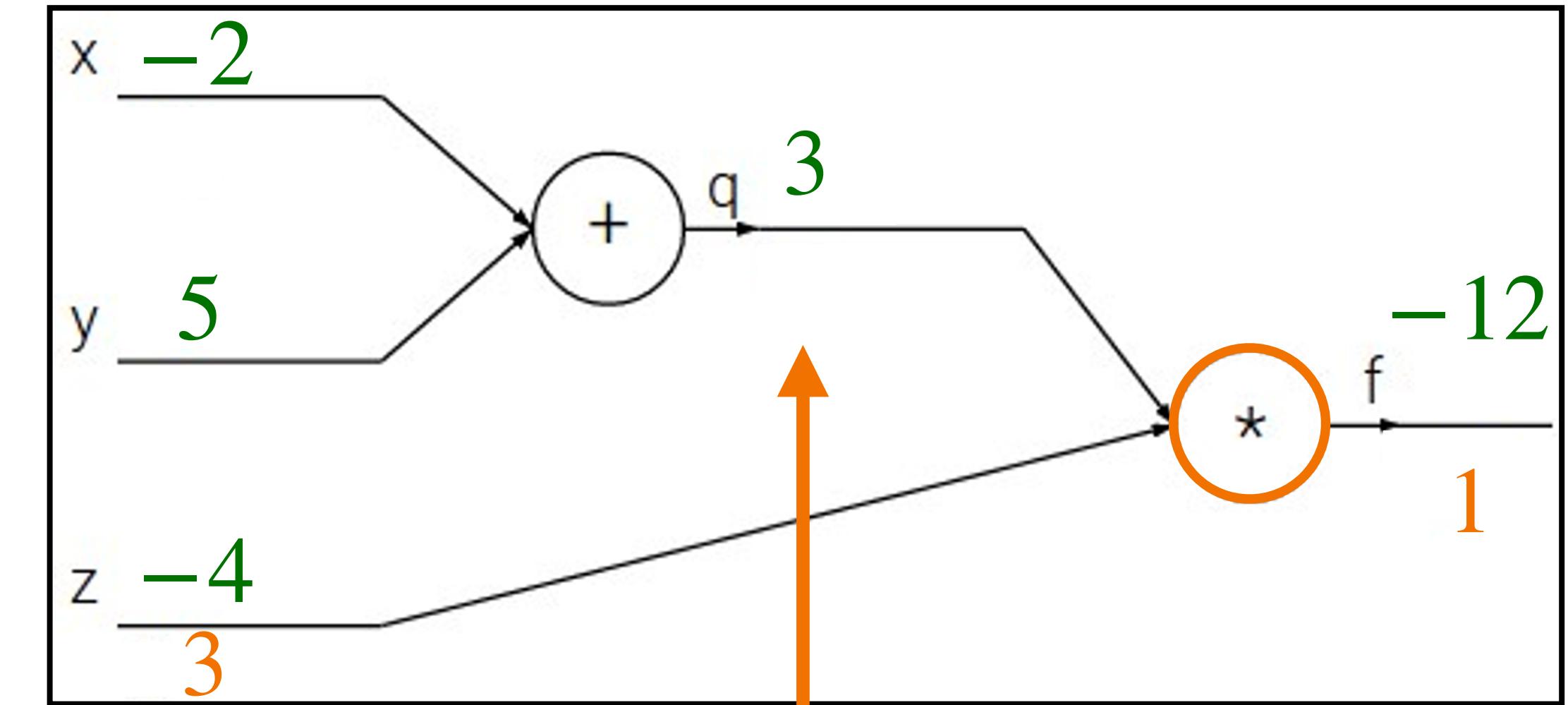
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

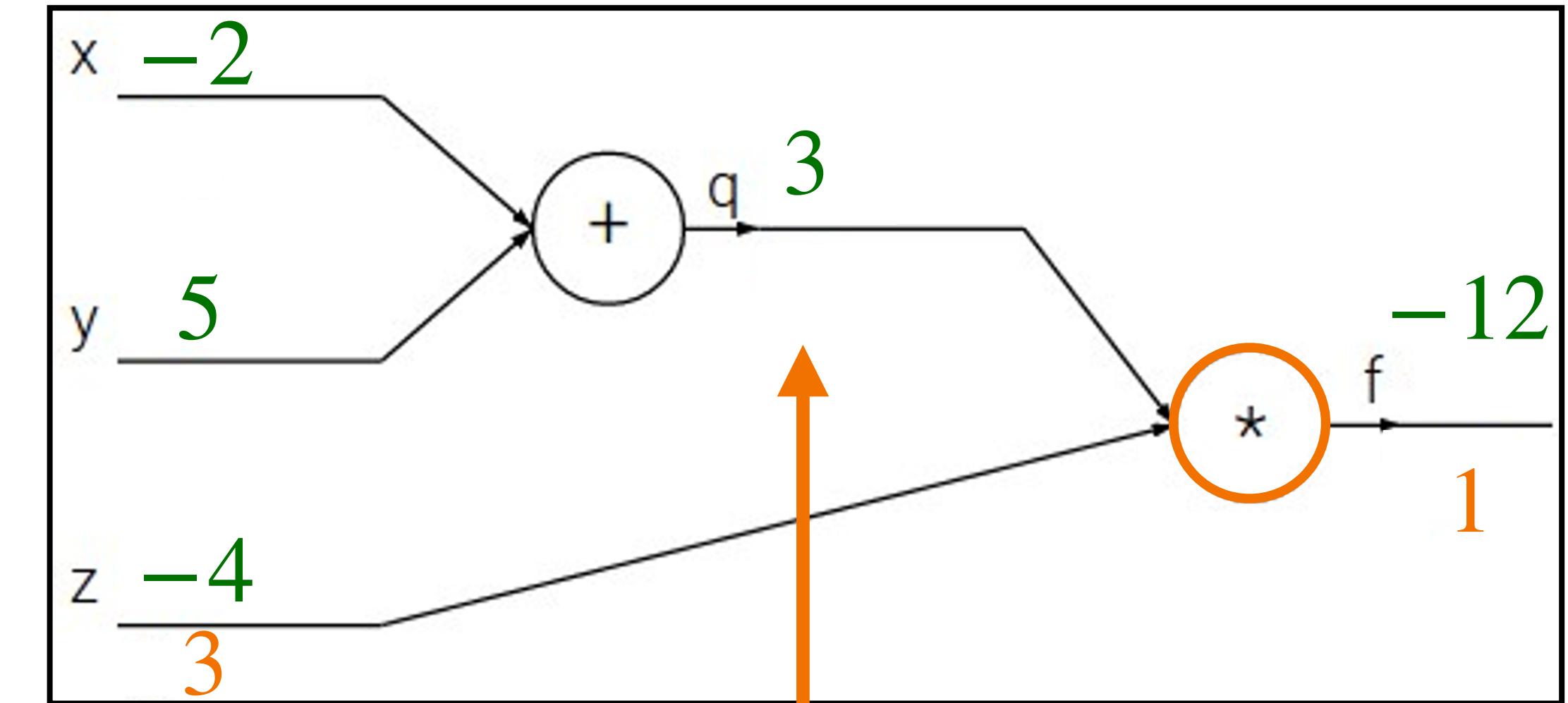
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

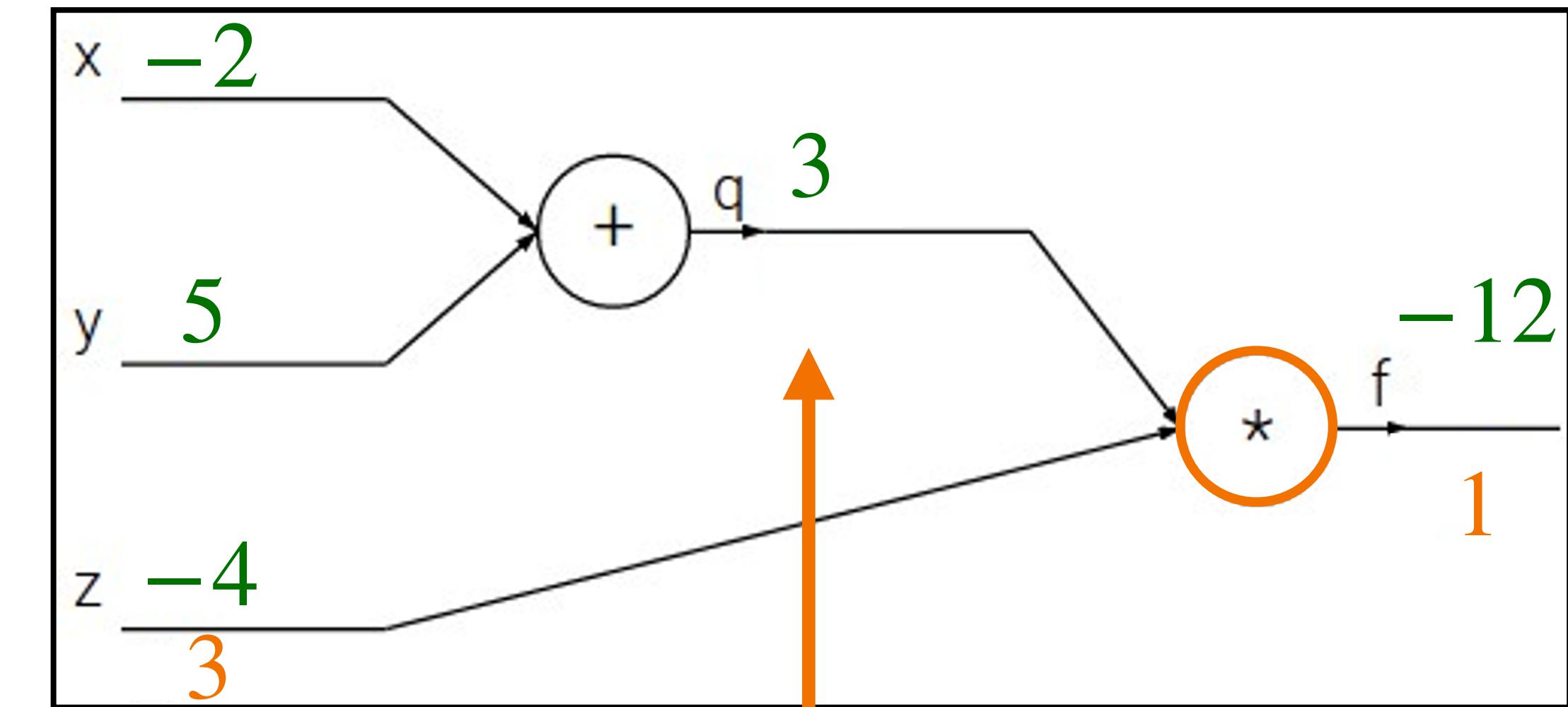
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q} = z$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

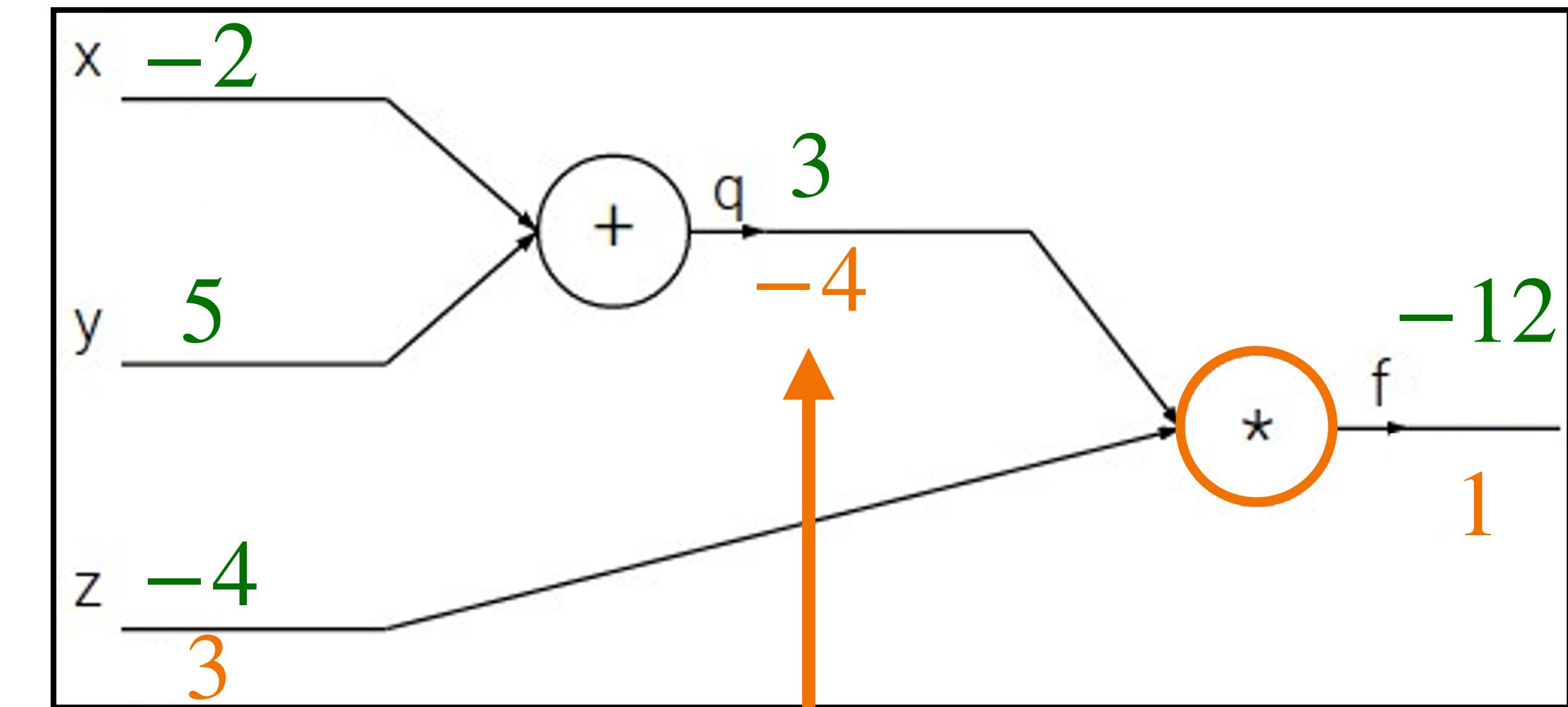
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q} = z$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

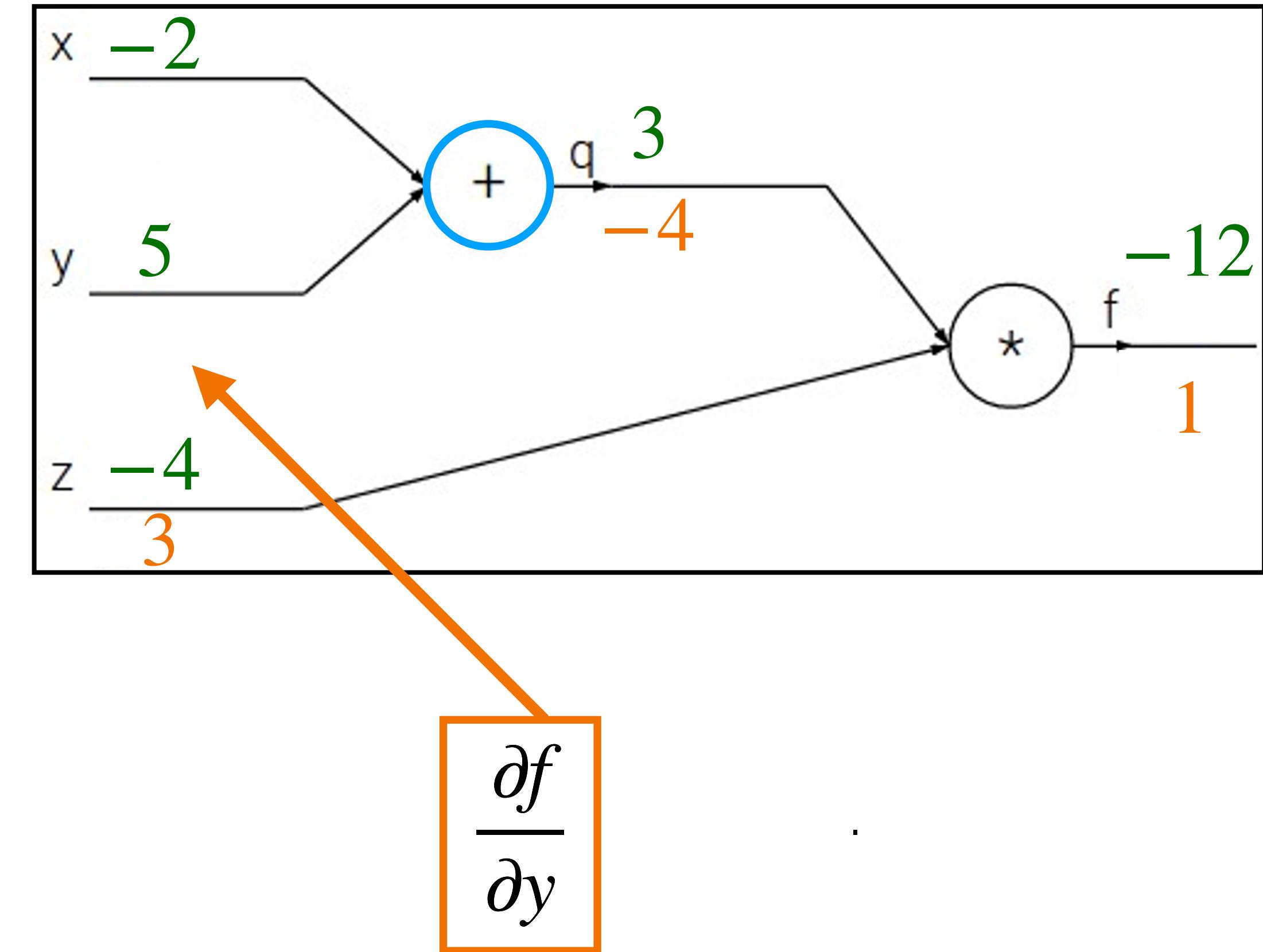
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

e.g. $x = -2, y = 5, z = -4$

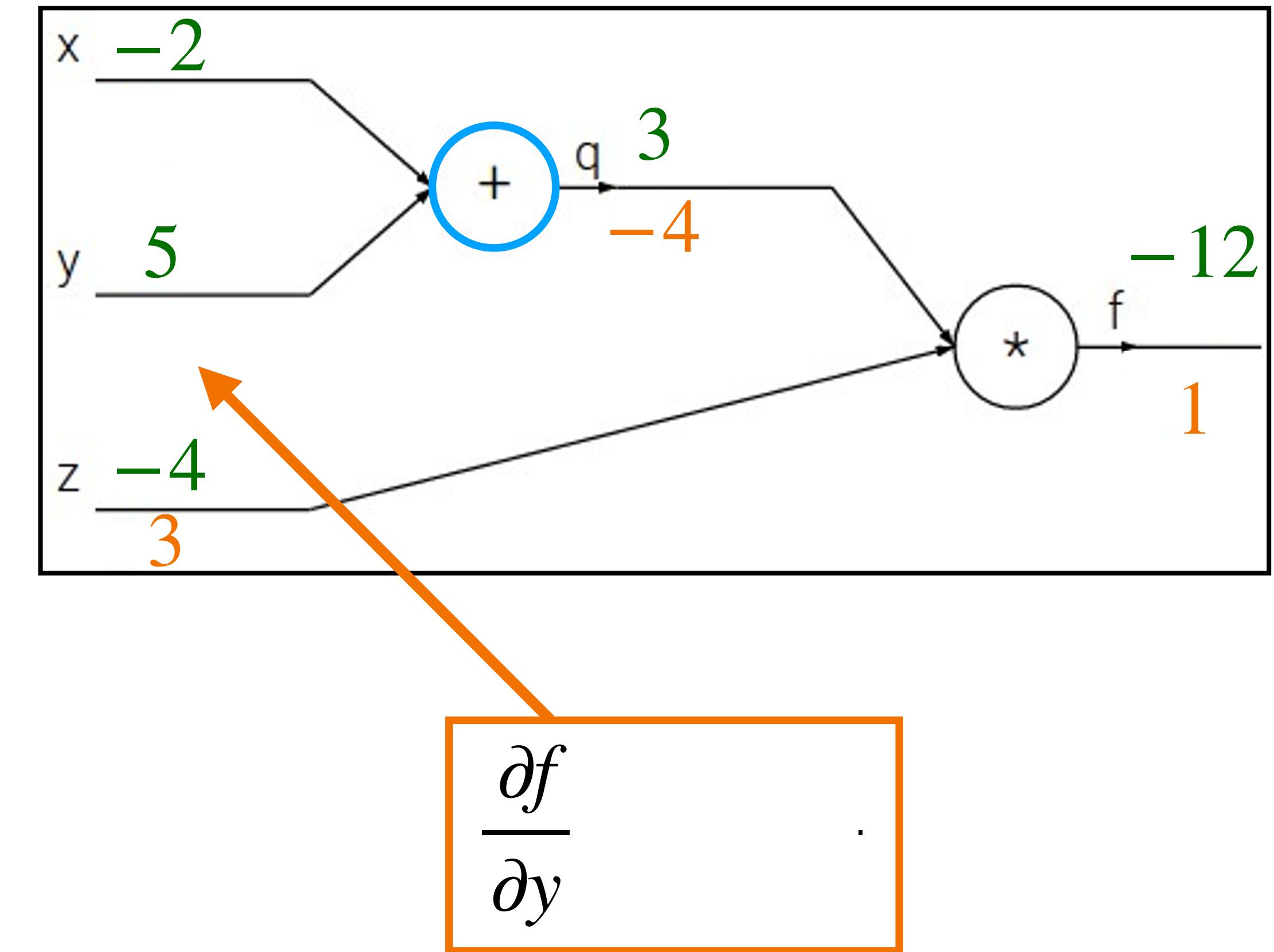
1. Forward pass: Compute outputs

$$q = x + y$$

$$f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

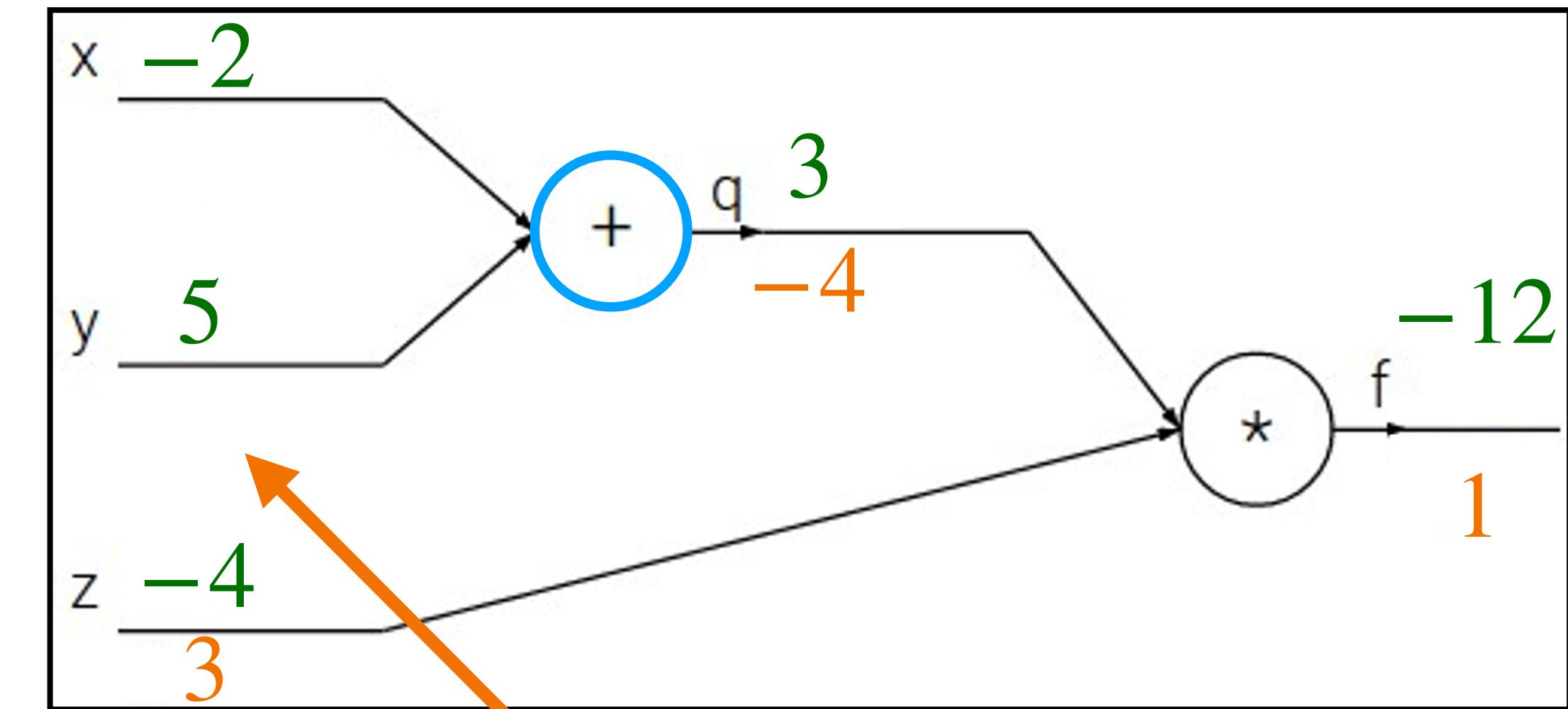
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

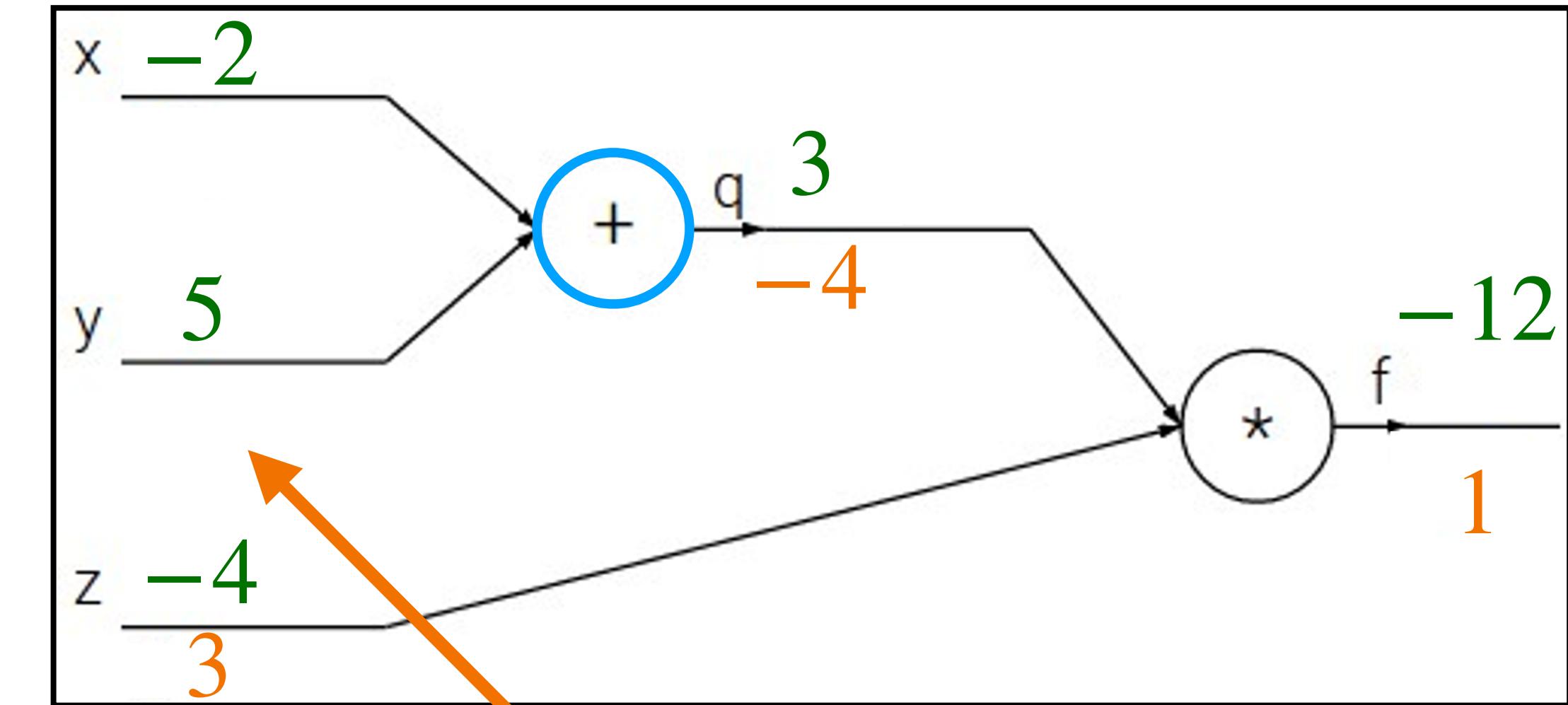
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

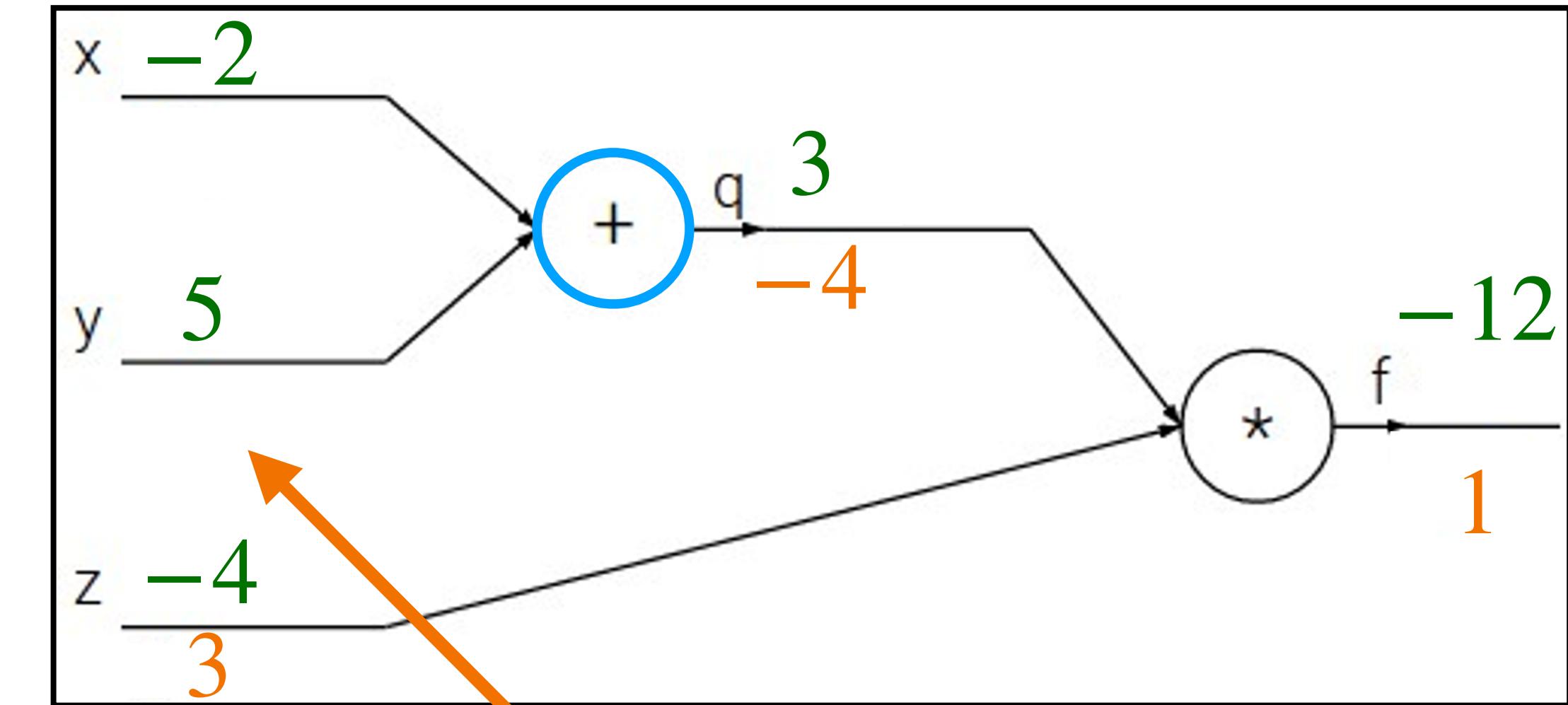
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

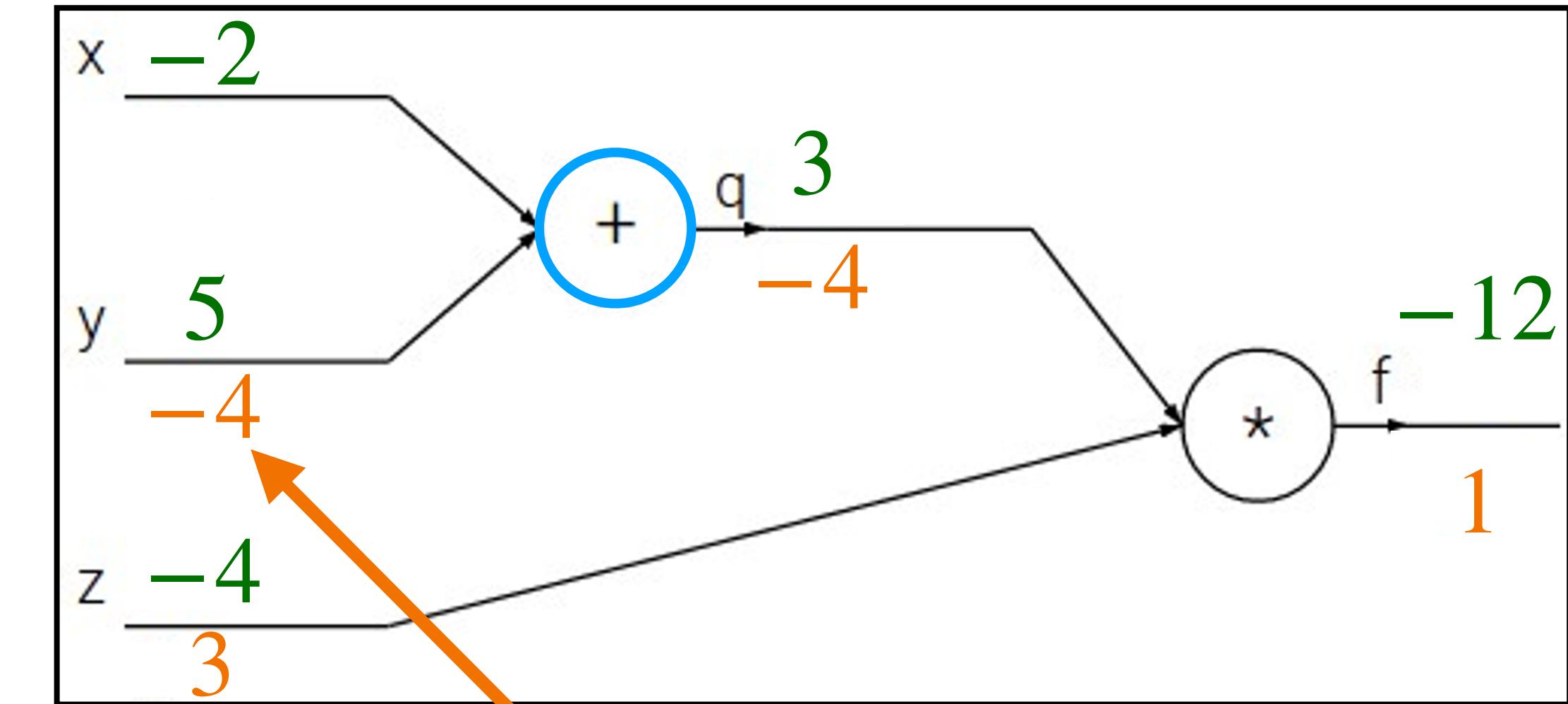
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial y} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

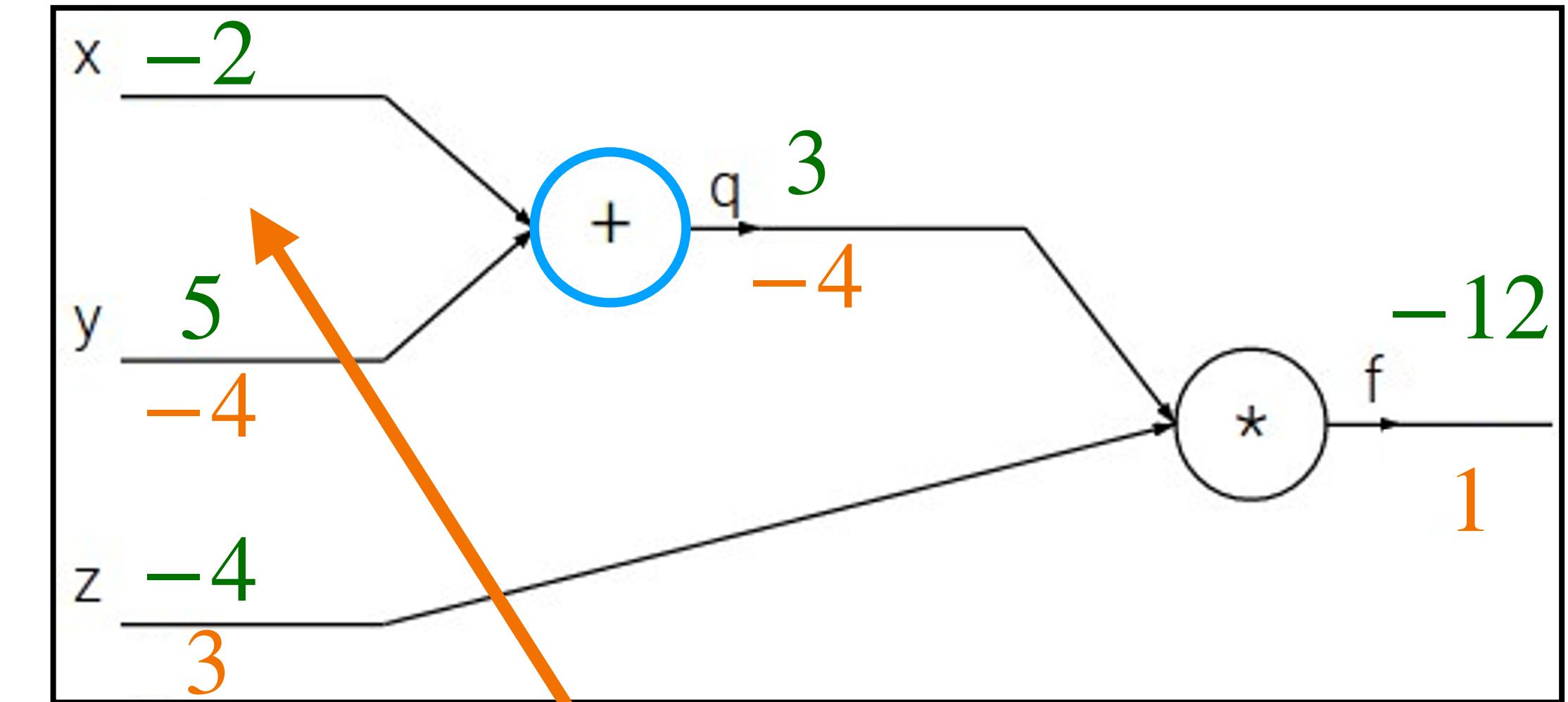
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

$$\frac{\partial q}{\partial x} = 1$$

Downstream
Gradient

Local
Gradient

Upstream
Gradient

Backpropagation: Simple Example

$$f(x, y, z) = (x + y) \cdot z$$

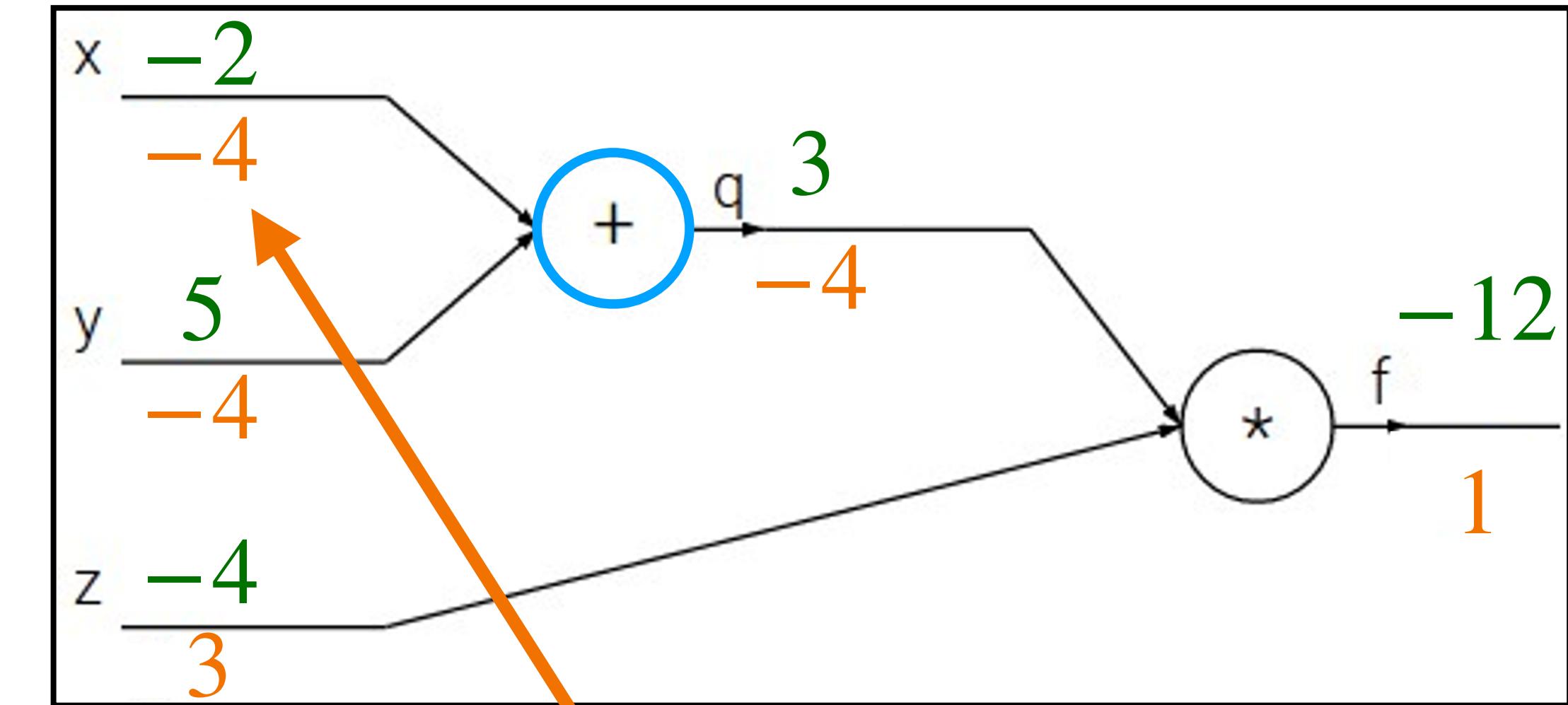
e.g. $x = -2, y = 5, z = -4$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x} \frac{\partial f}{\partial q}$$

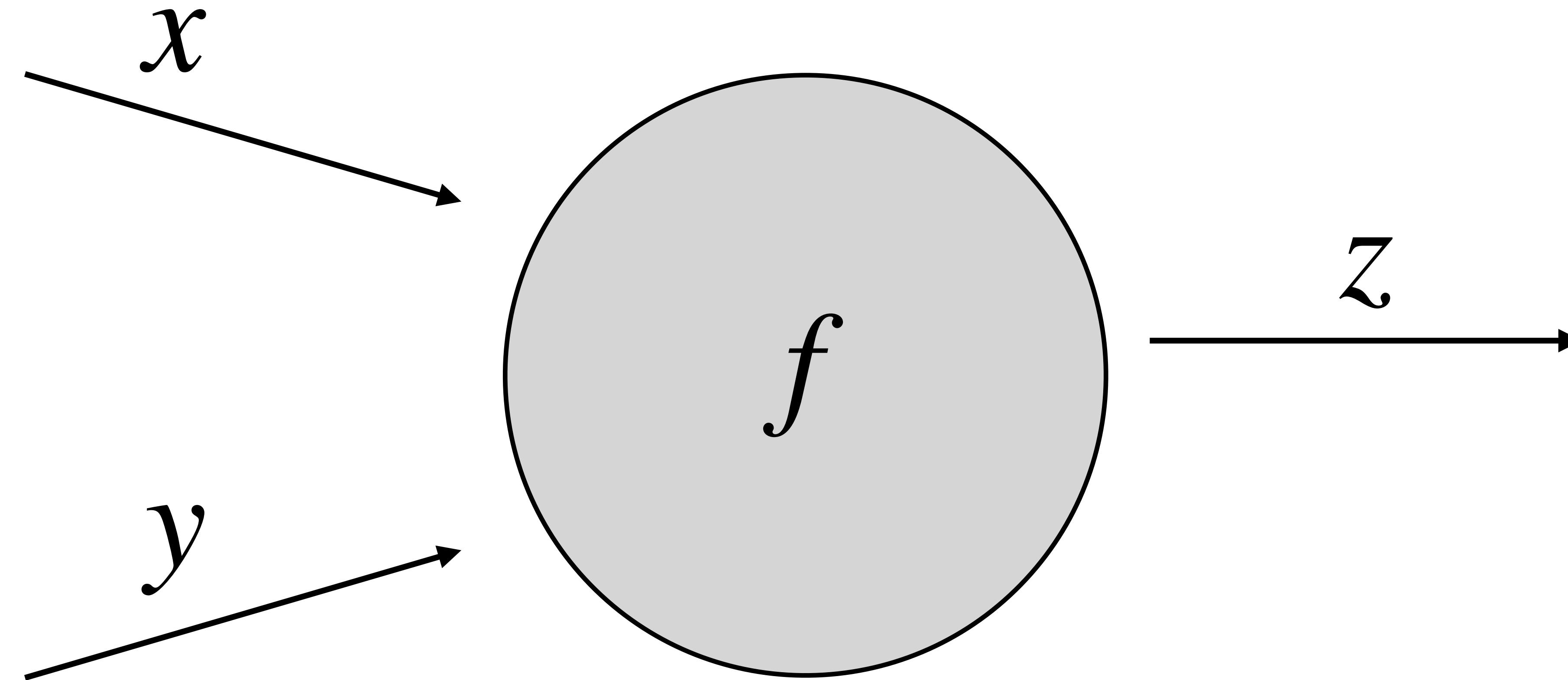
$$\frac{\partial q}{\partial x} = 1$$

Downstream
Gradient

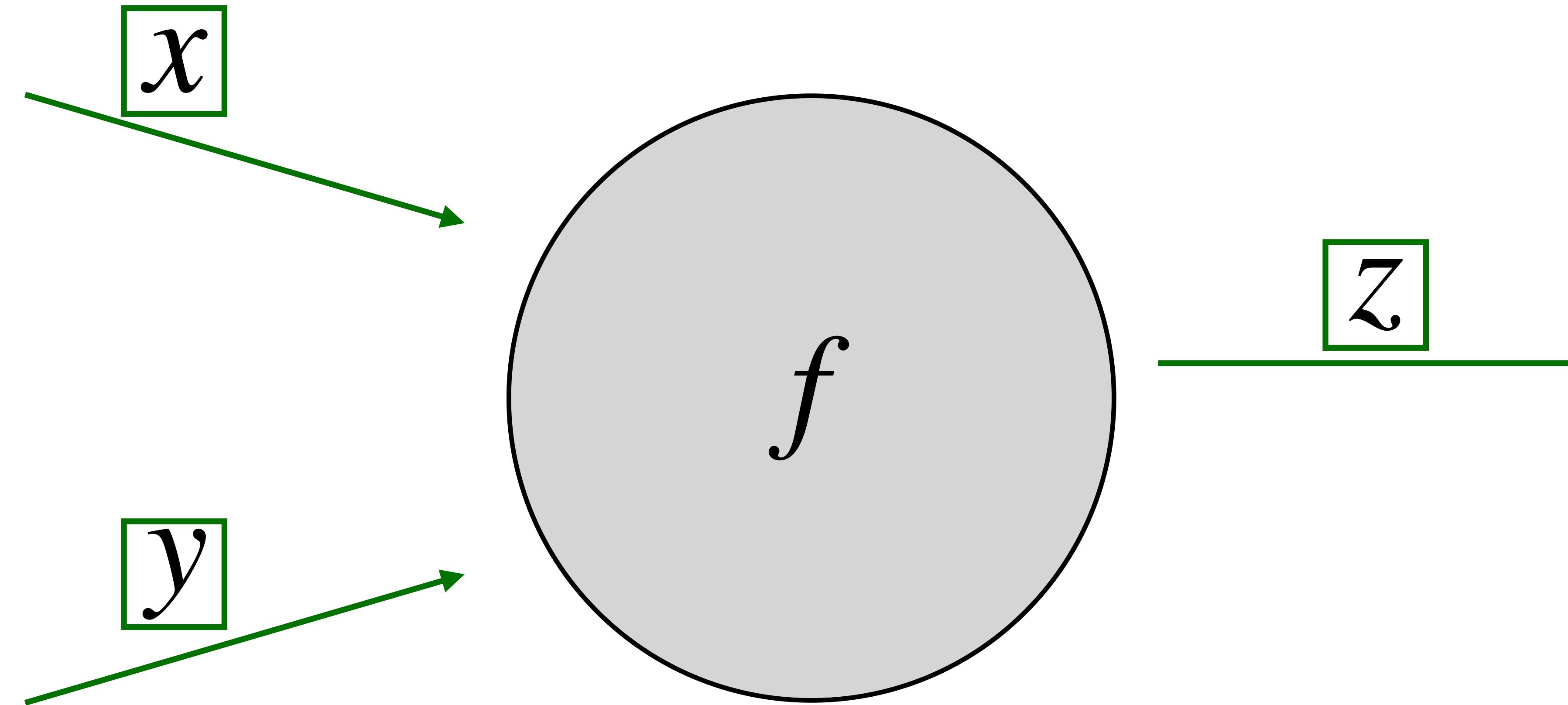
Local
Gradient

Upstream
Gradient

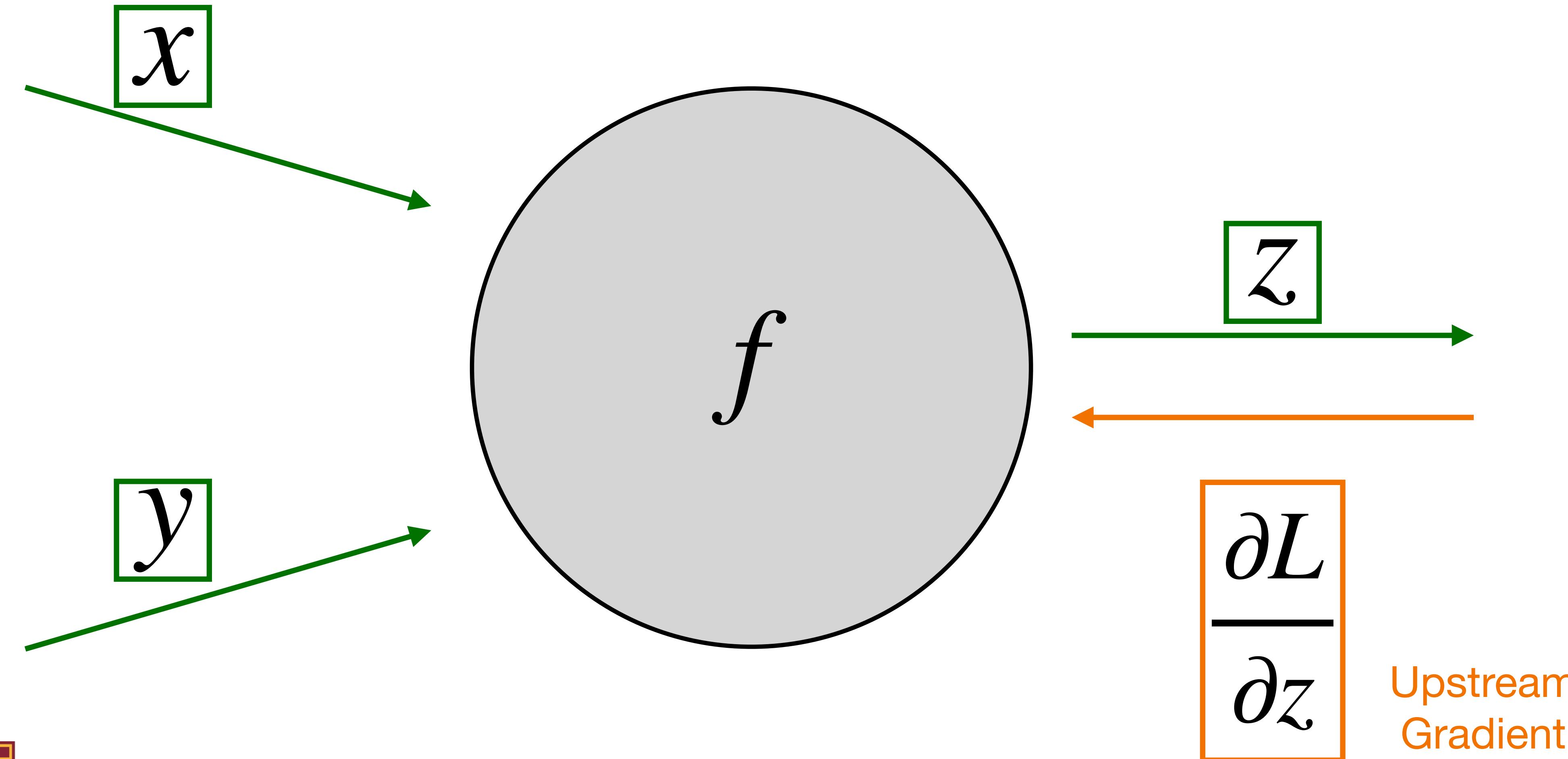
Local Properties of Backpropagation



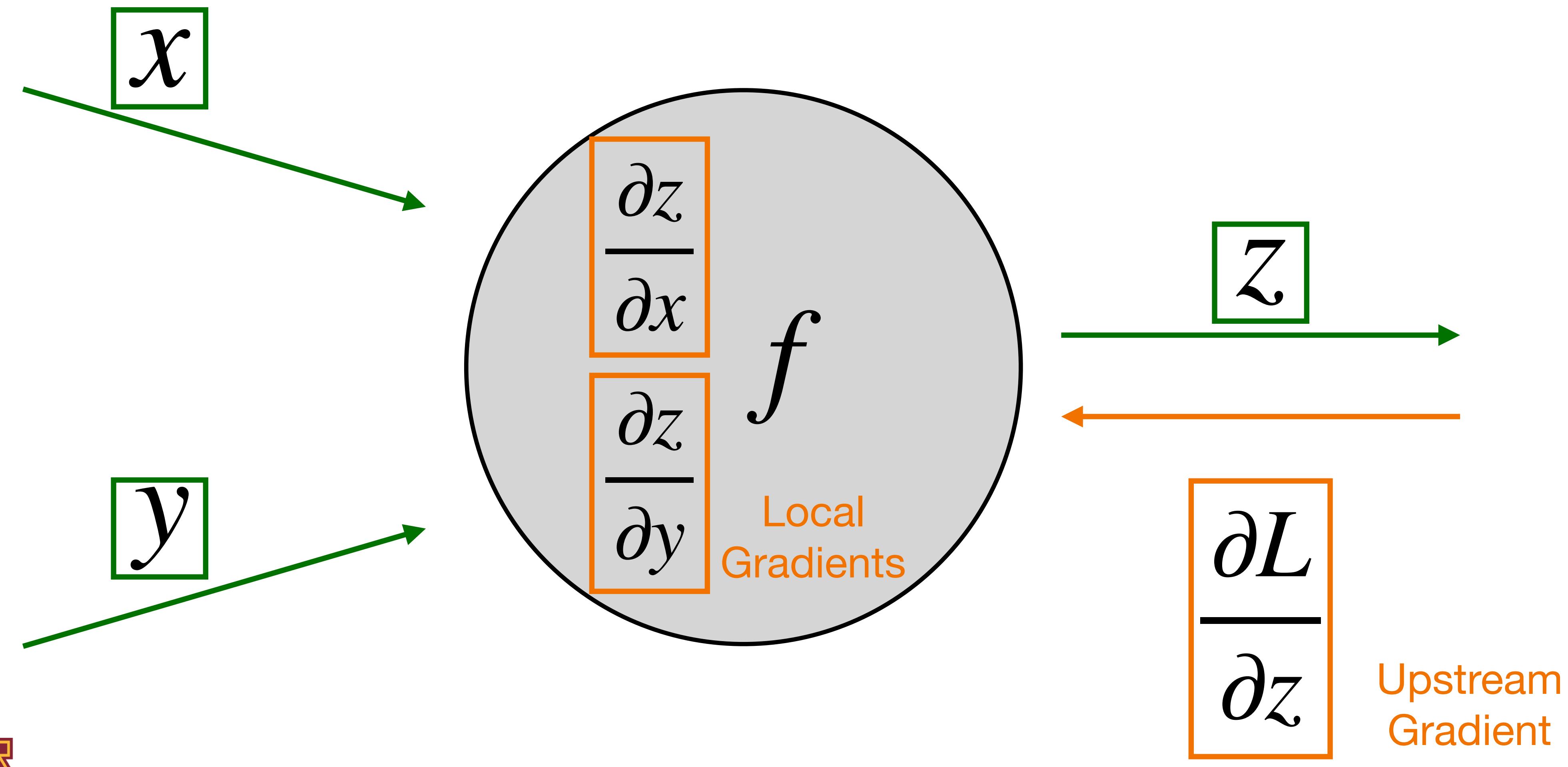
Local Properties of Backpropagation



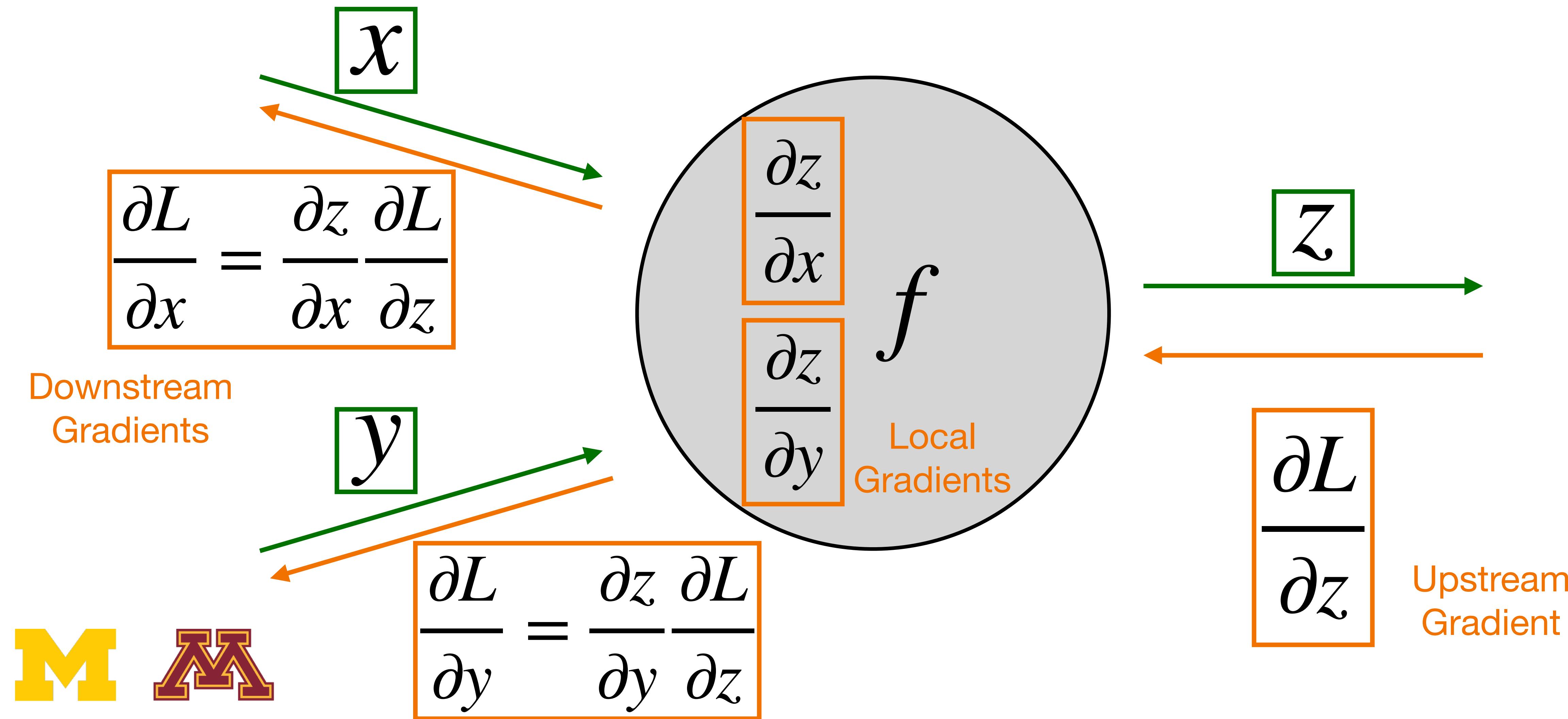
Local Properties of Backpropagation



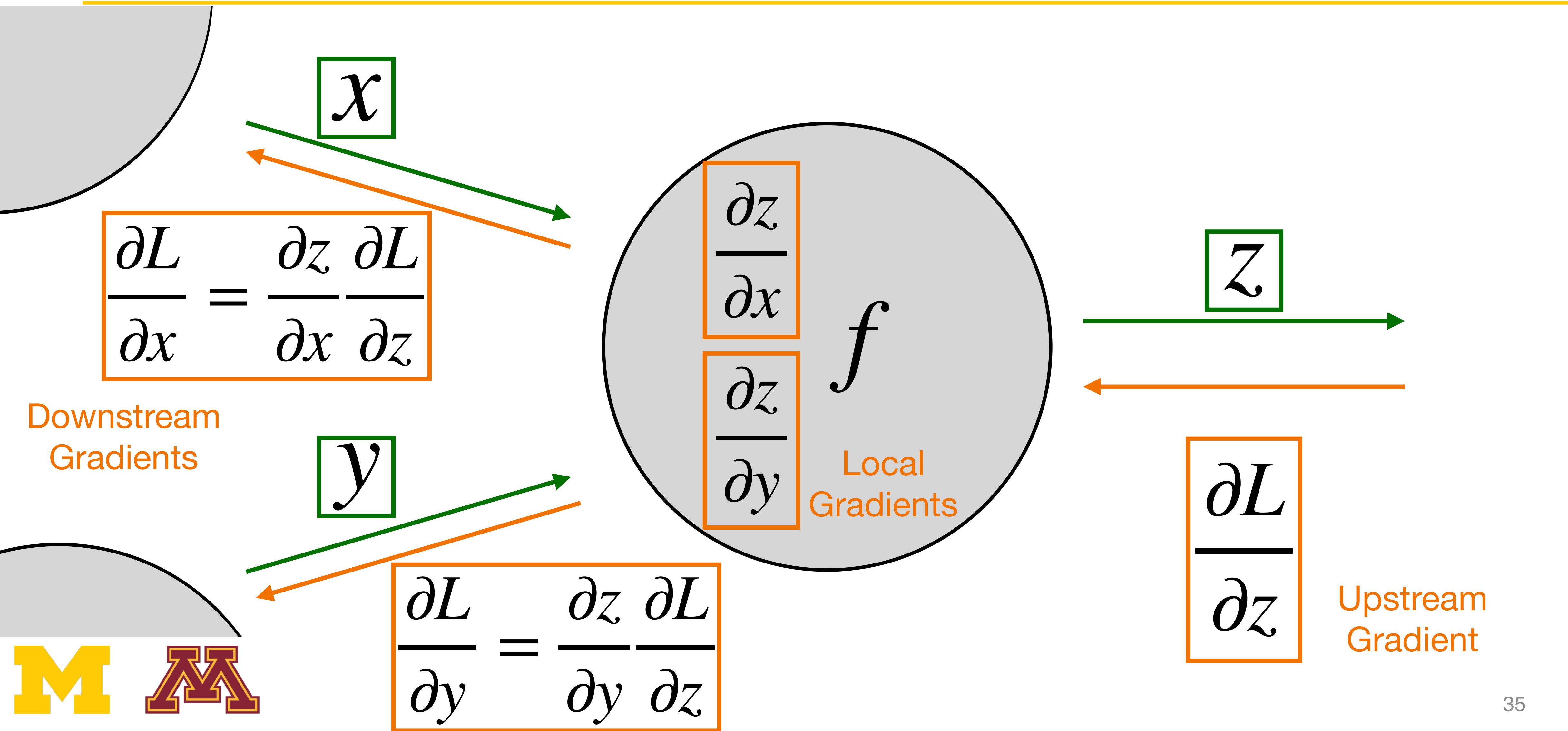
Local Properties of Backpropagation



Local Properties of Backpropagation



Local Properties of Backpropagation

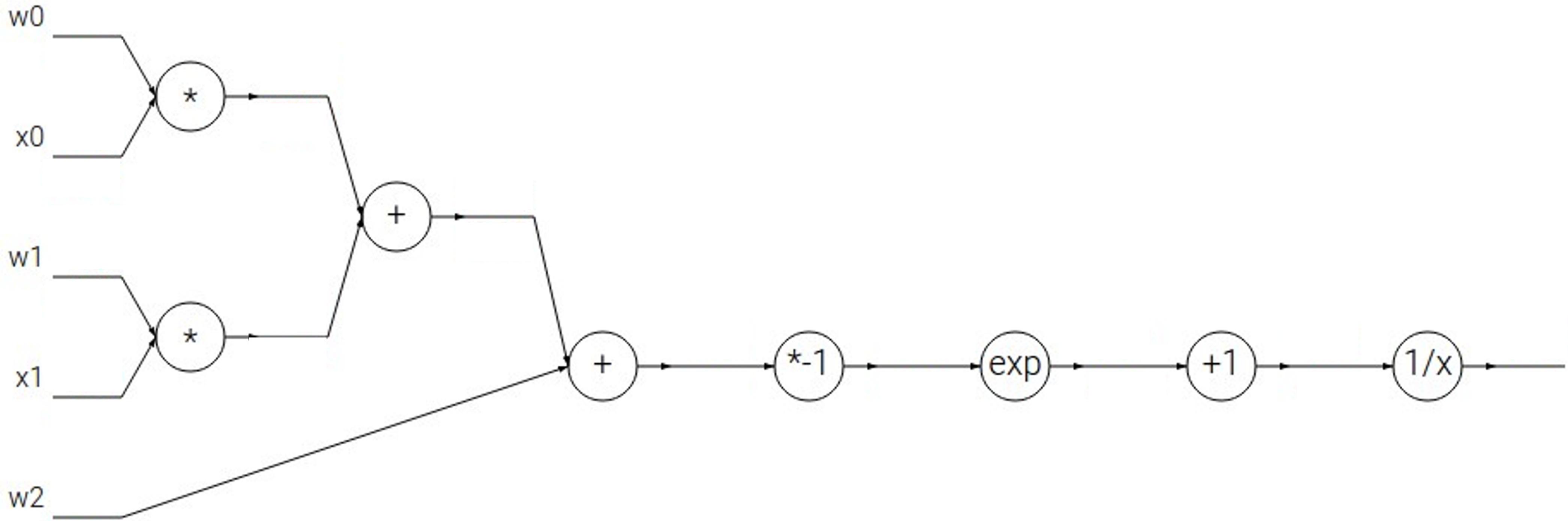


Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

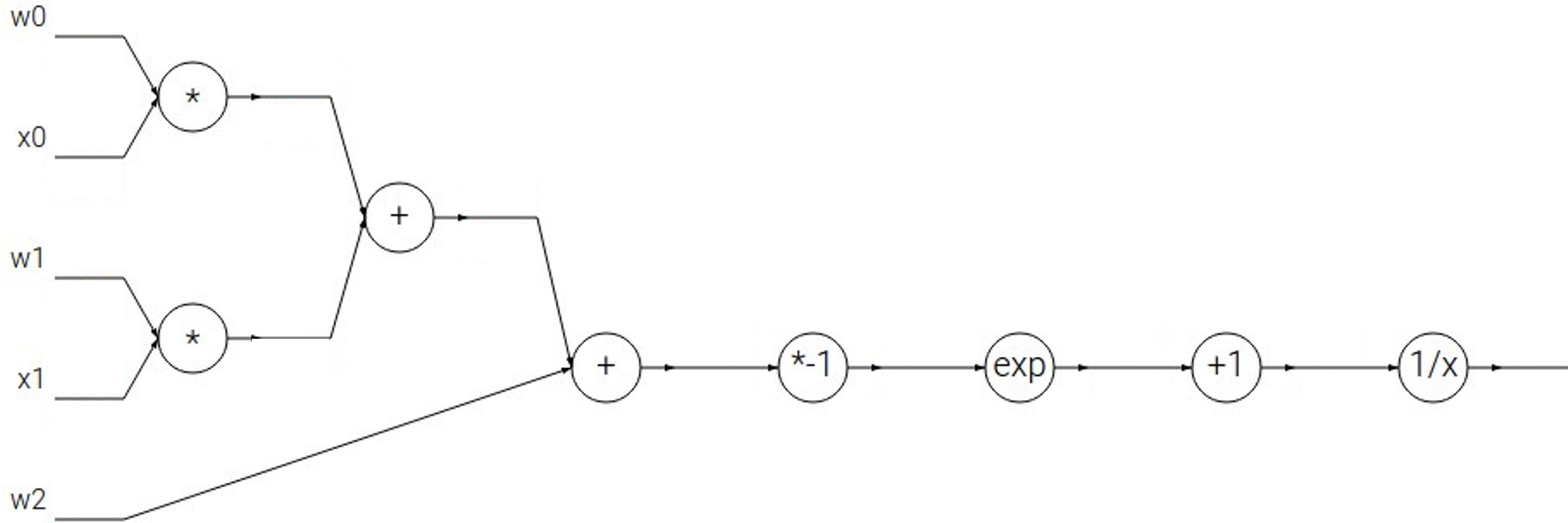
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

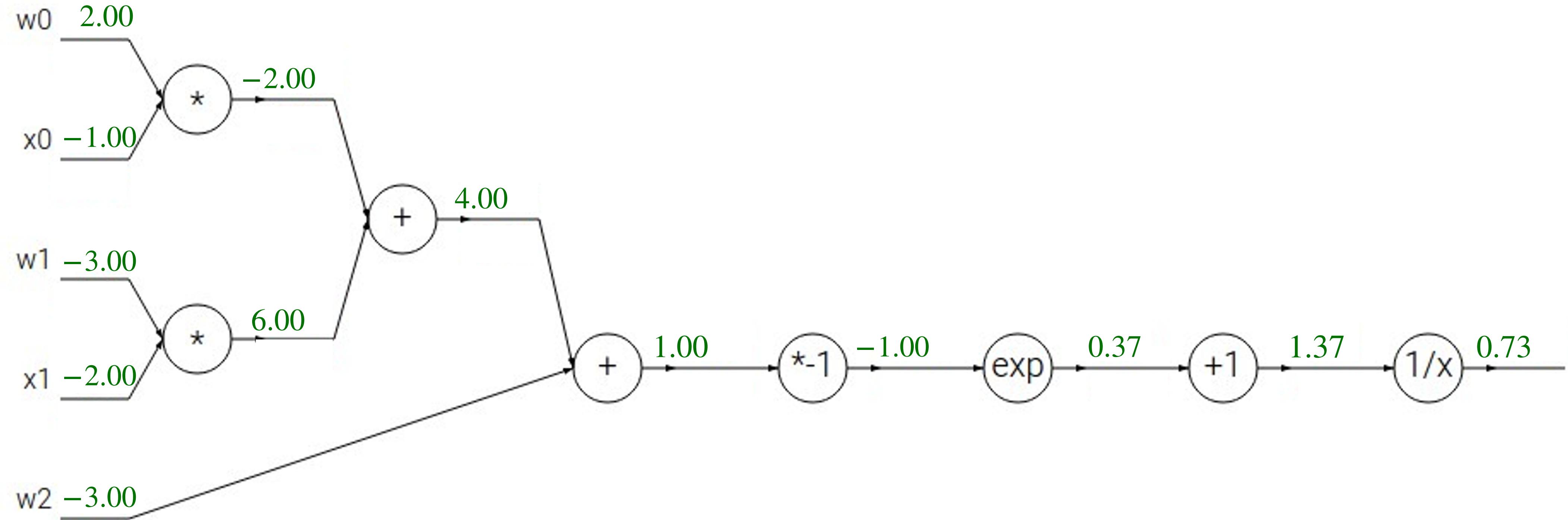
$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

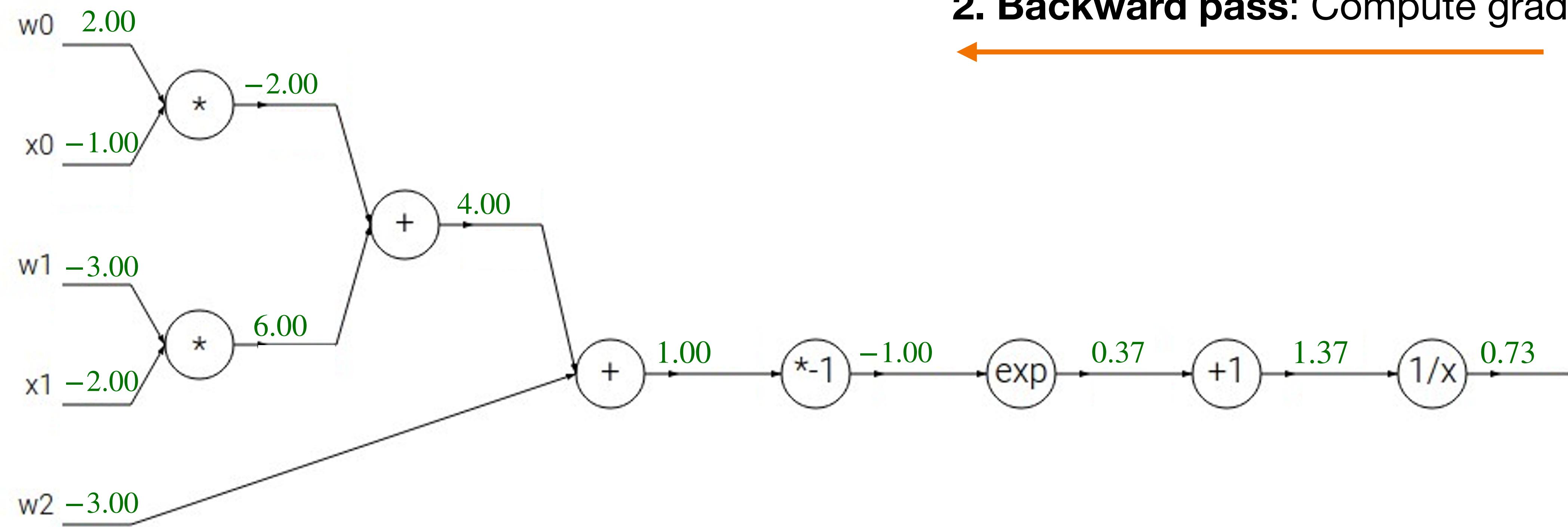
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

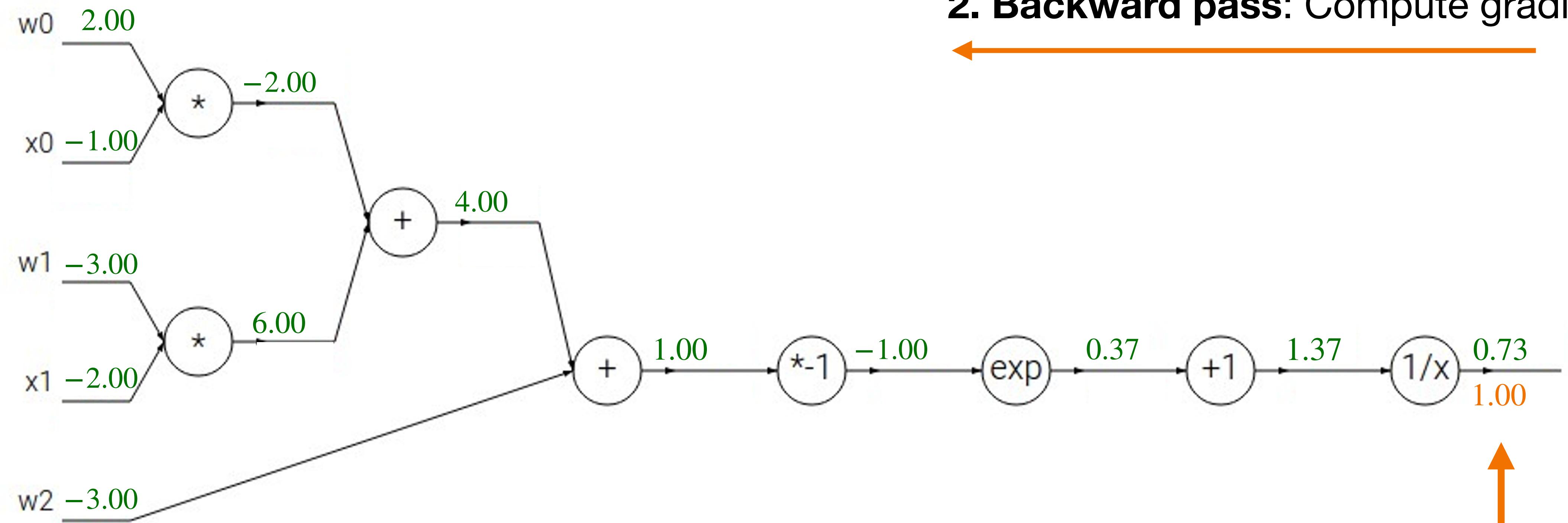


2. Backward pass: Compute gradients



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Base Case

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

```

graph LR
    x0[2.00] --> M1(( ))
    x1[-1.00] --> M1
    M1 -- "*" --> P1(( ))
    w0[2.00] --> P1
    P1 -- "-" --> S1(( ))
    S1 -- "+" --> P2(( ))
    w1[-3.00] --> P2
    P2 -- "*" --> M2(( ))
    x1[-2.00] --> M2
    M2 -- "6.00" --> P3(( ))
    P3 -- "*" --> S2(( ))
    S2 -- "+" --> P4(( ))
    w2[-3.00] --> P4
    P4 -- "1.00" --> M3(( ))
    M3 -- "-" --> P5(( ))
    P5 -- "1.00" --> M4(( ))
    M4 -- "0.37" --> EXP((exp))
    EXP -- "0.37" --> M5(( ))
    M5 -- "-1.00" --> M6(( ))
    M6 -- "*" --> M7(( ))
    M7 -- "-1" --> M8(( ))
    M8 -- "+" --> M9(( ))
    M9 -- "0.37" --> M10(( ))
    M10 -- "+1" --> EXP
    EXP -- "0.73" --> M11(( ))
    M11 -- "0.73" --> M12(( ))
    M12 -- "1.00" --> ONE_X((1/x))
    ONE_X -- "1.00" --> OUT(( ))
  
```

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Local Gradient

$$\frac{\partial}{\partial x} \left[\frac{1}{x} \right] = -\frac{1}{x^2}$$

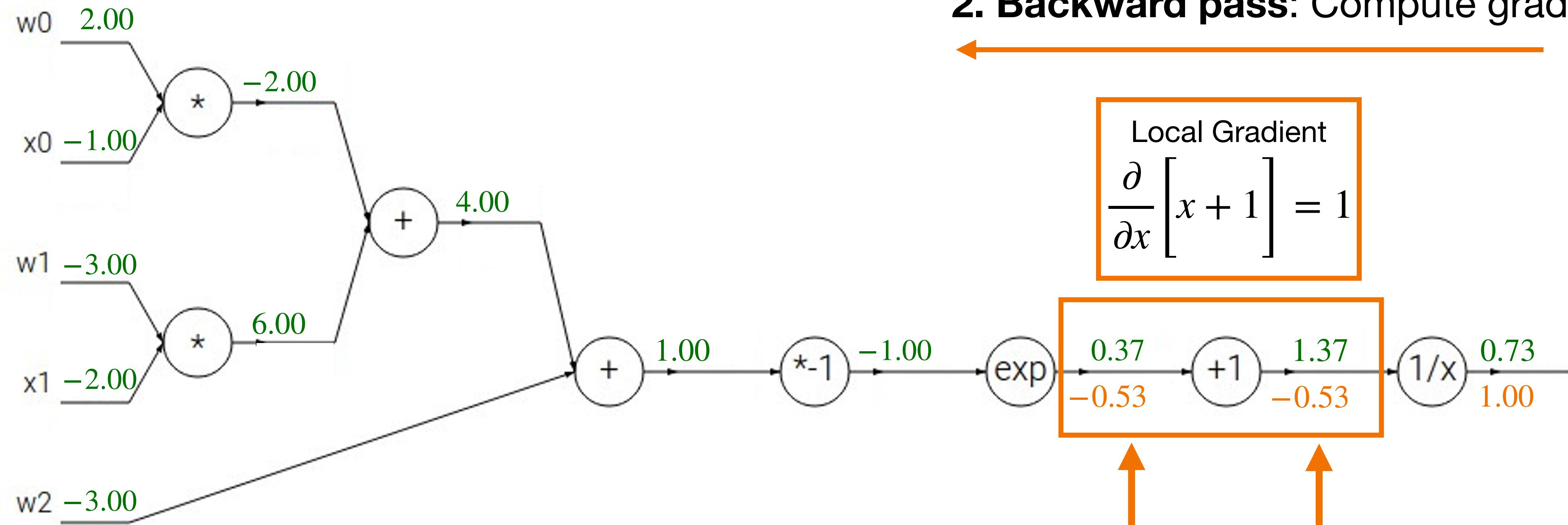
$$\begin{matrix} 1.37 \\ -0.53 \end{matrix} \quad \begin{matrix} 0.73 \\ 1.00 \end{matrix}$$

Downstream
Gradient

Upstream
Gradient

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Local Gradient

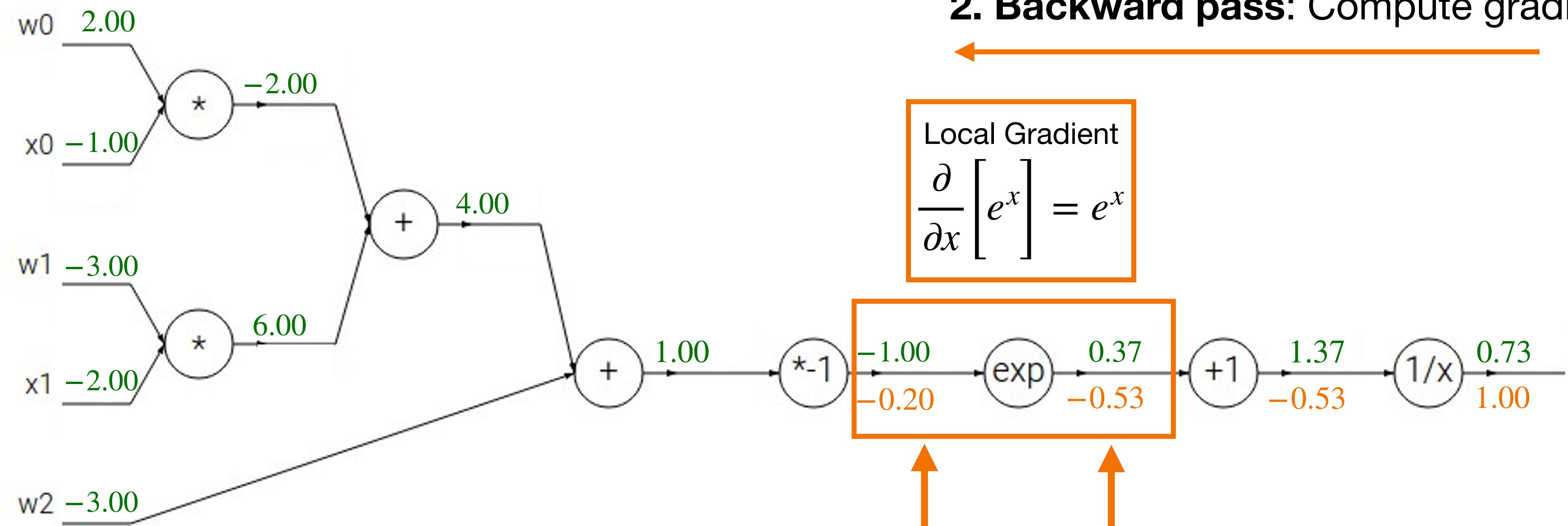
$$\frac{\partial}{\partial x} [x + 1] = 1$$

Downstream Gradient

Upstream Gradient

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$\begin{matrix} w_0 & 2.00 \\ x_0 & -1.00 \end{matrix}$$

$$\begin{matrix} w_1 & -3.00 \\ x_1 & -2.00 \end{matrix}$$

$$\begin{matrix} w_2 & -3.00 \end{matrix}$$

Local Gradient

$$\frac{\partial}{\partial x} [-1 \cdot x] = -1$$

$$\begin{matrix} 1.00 & -1.00 \\ 0.20 & -0.20 \end{matrix}$$

Downstream
Gradient

1. Forward pass: Compute outputs

2. Backward pass: Compute gradients



$$\begin{matrix} *-1 & -0.20 \\ 0.37 & -0.53 \end{matrix}$$

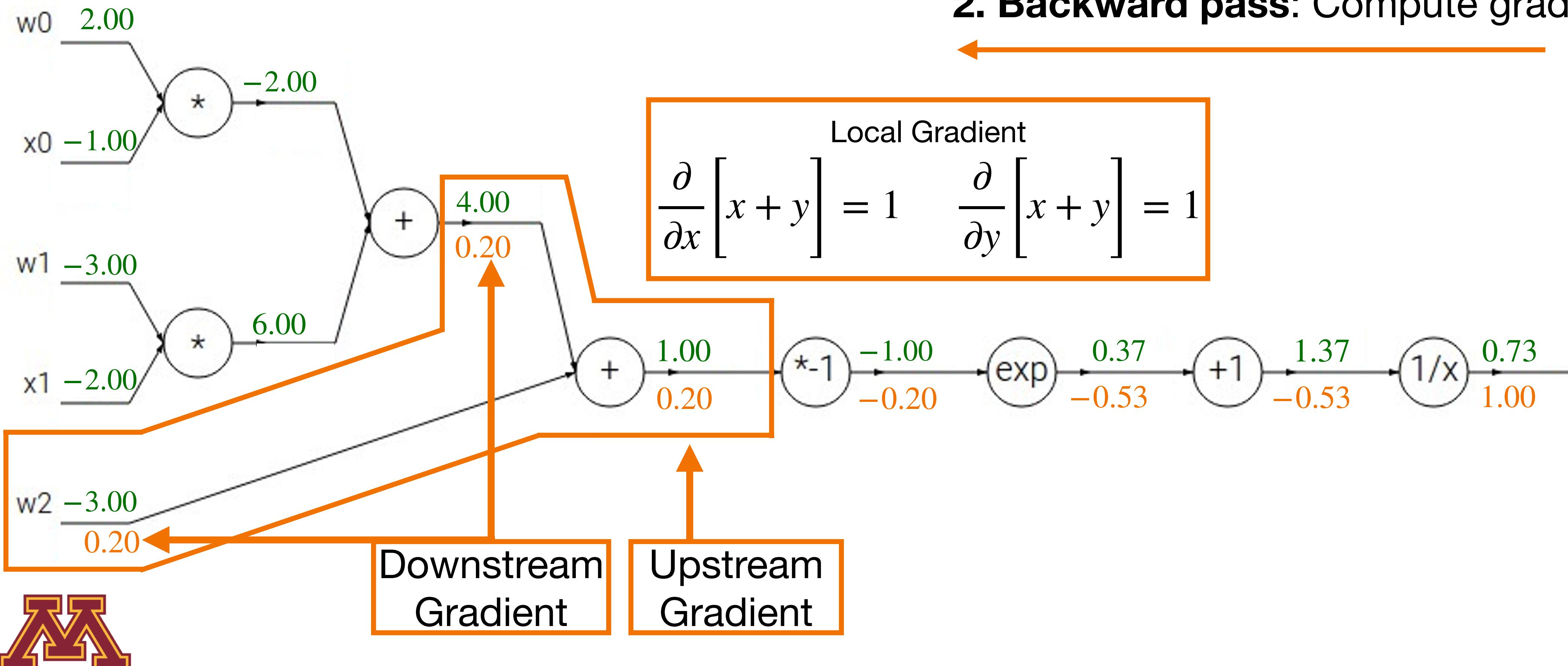
Upstream
Gradient

$$\begin{matrix} \exp & -0.53 \\ +1 & -0.53 \end{matrix}$$

$$\begin{matrix} 1/x & 0.73 \\ 1.00 & 1.00 \end{matrix}$$

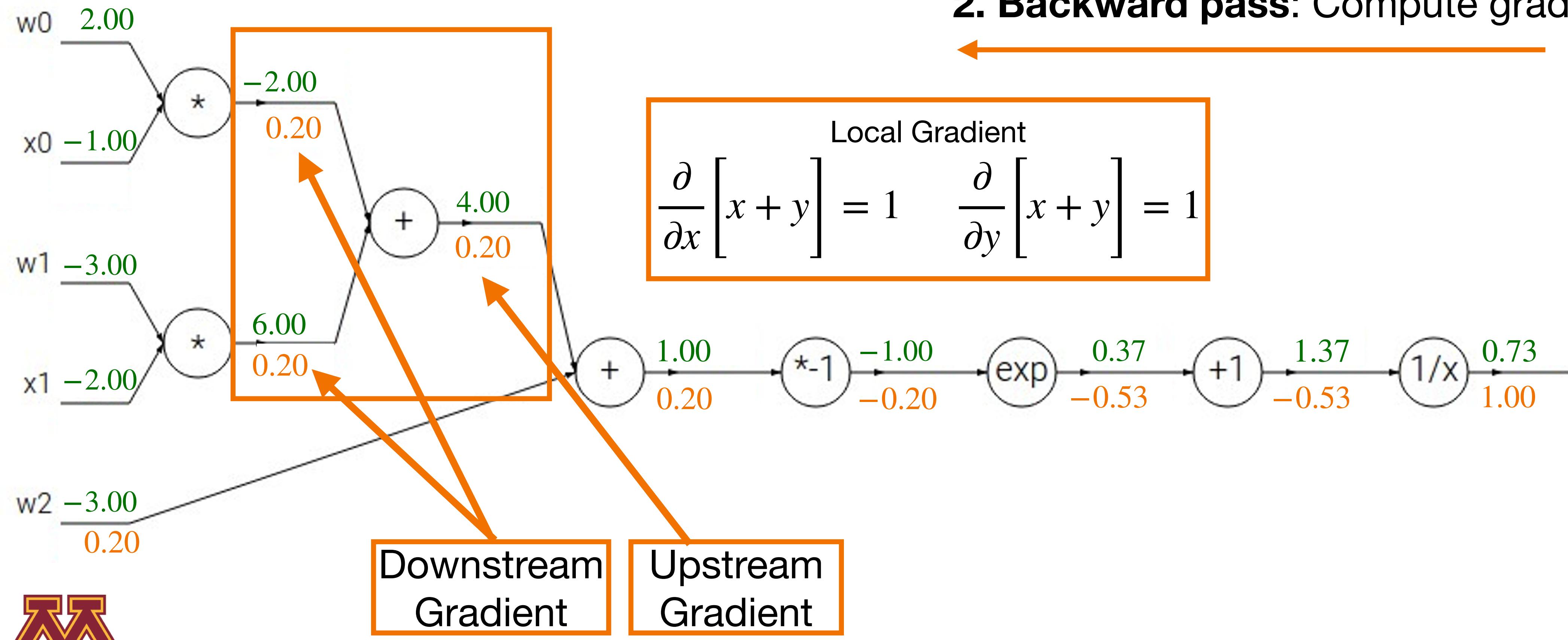
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



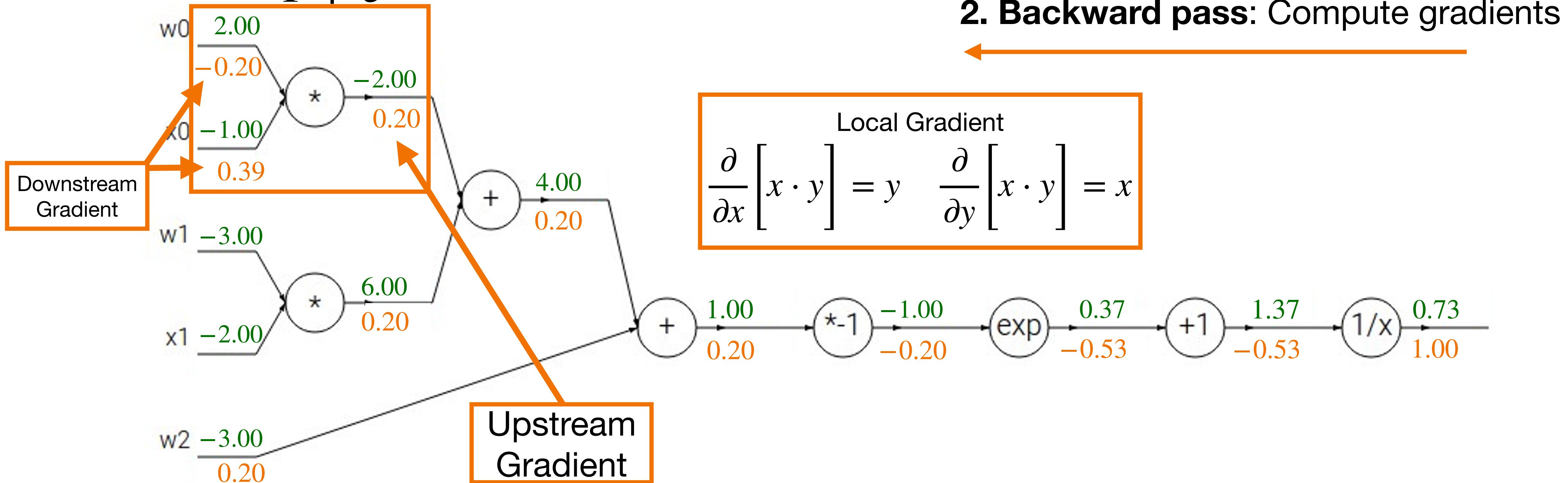
Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

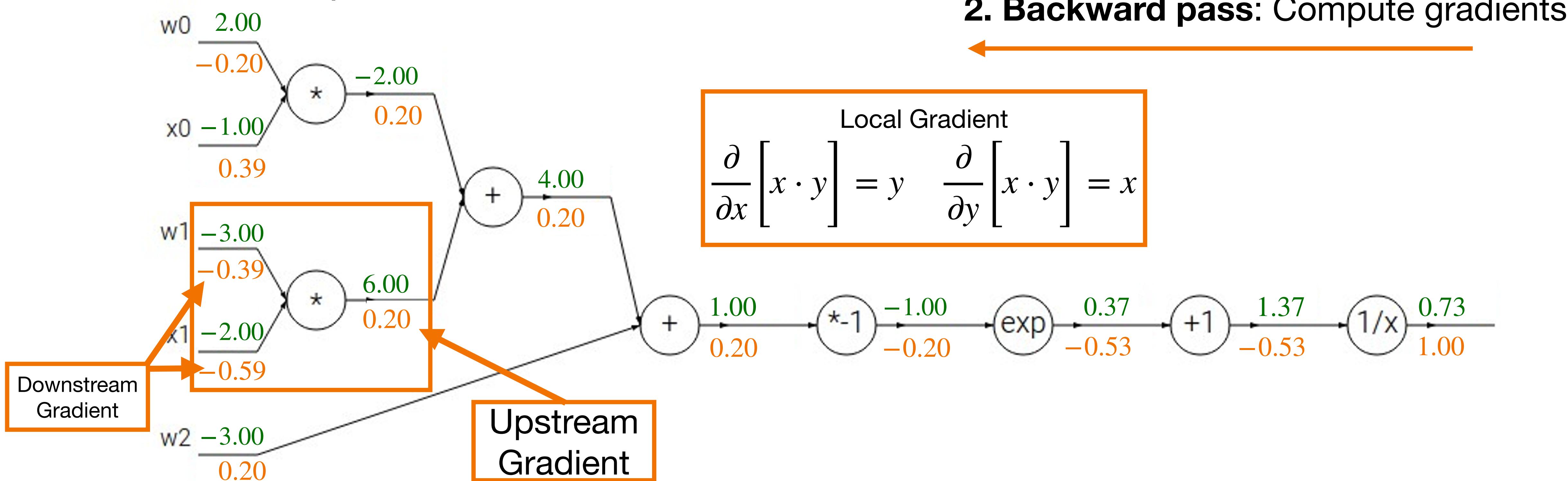


1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Another example

$$f(x, w) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

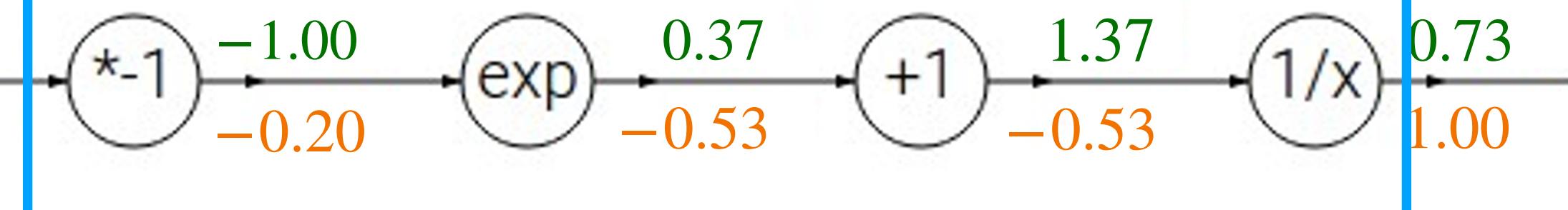
$= \sigma(w_0x_0 + w_1x_1 + w_2)$

```

graph LR
    x0[1.00] --> M1(( ))
    x1[-2.00] --> M1
    w0[2.00] --> M1
    w1[-3.00] --> M1
    w2[-3.00] --> M2(( ))
    M1 -- "0.39" --> P1(( ))
    M1 -- "0.20" --> P2(( ))
    P1 -- "4.00" --> S(( ))
    P2 -- "0.20" --> S
    S -- "0.20" --> O(( ))
    
```

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid

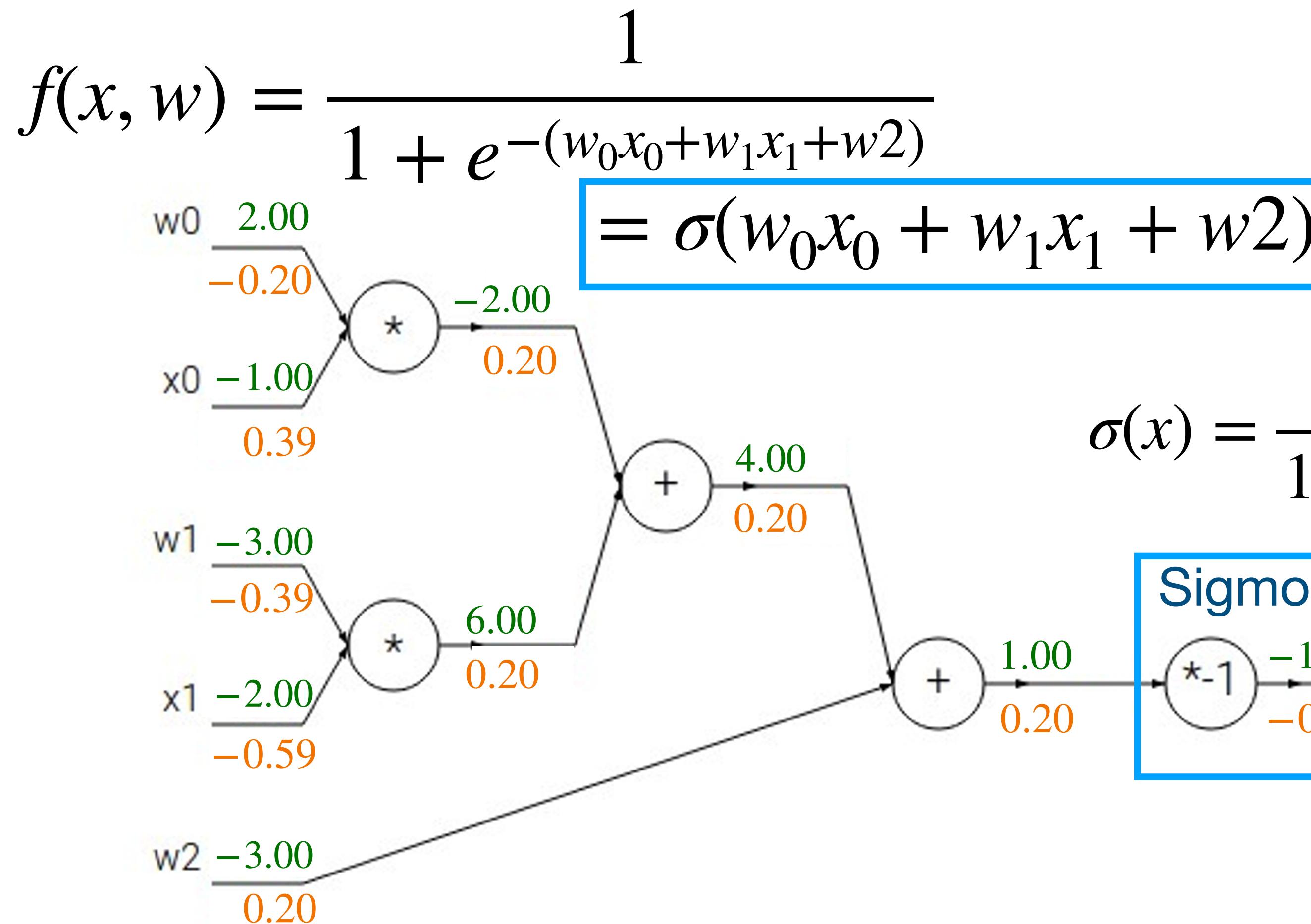


1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Computational graph is not unique: we can use primitives that have simple local gradients

Another example



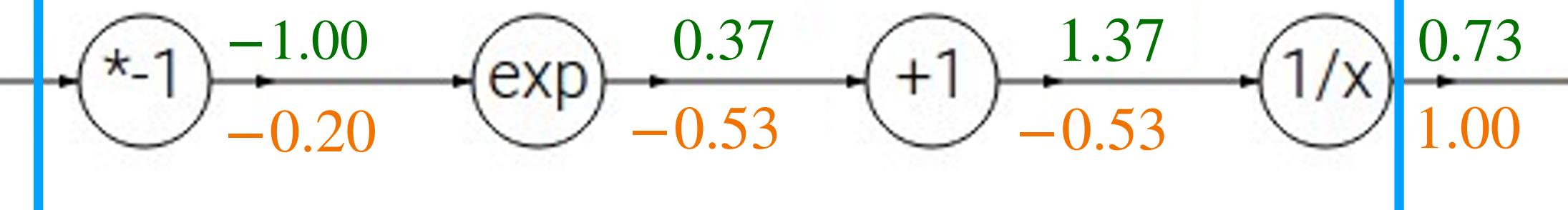
1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Computational graph is not unique: we can use primitives that have simple local gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

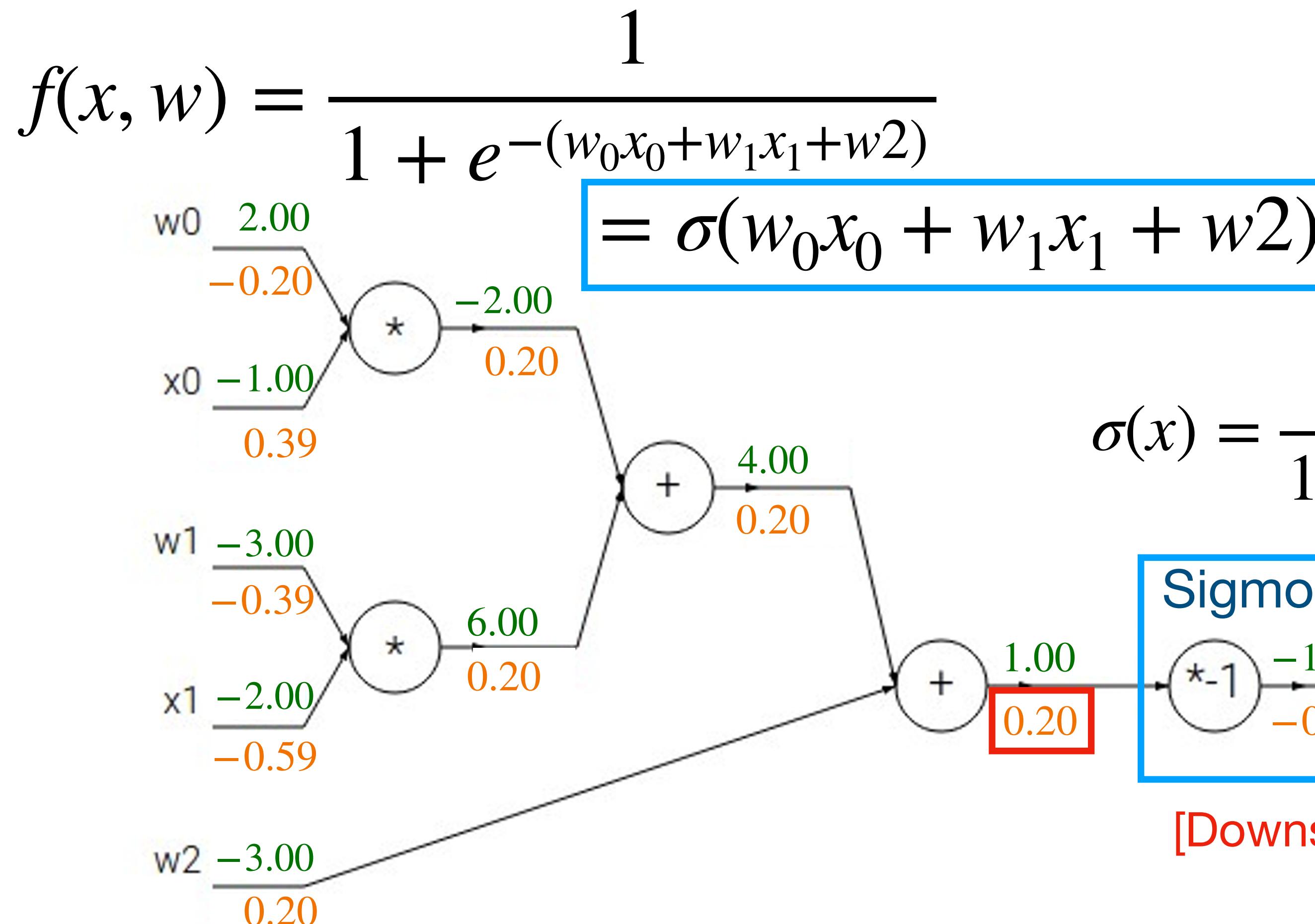
Sigmoid



Sigmoid local gradient:

$$\frac{\partial}{\partial x} [\sigma(x)] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

Another example



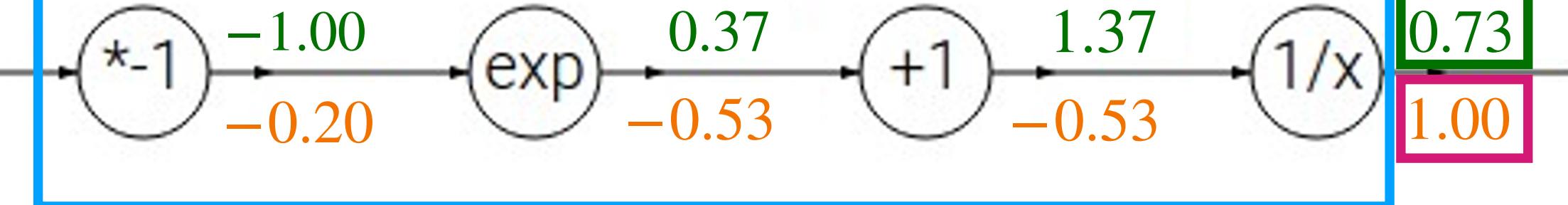
1. Forward pass: Compute outputs

2. Backward pass: Compute gradients

Computational graph is not unique: we can use primitives that have simple local gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid



[Downstream] = [Local] · [Upstream]

$$= (1 - 0.73) \cdot 0.73 \cdot 1.00 = 0.20$$

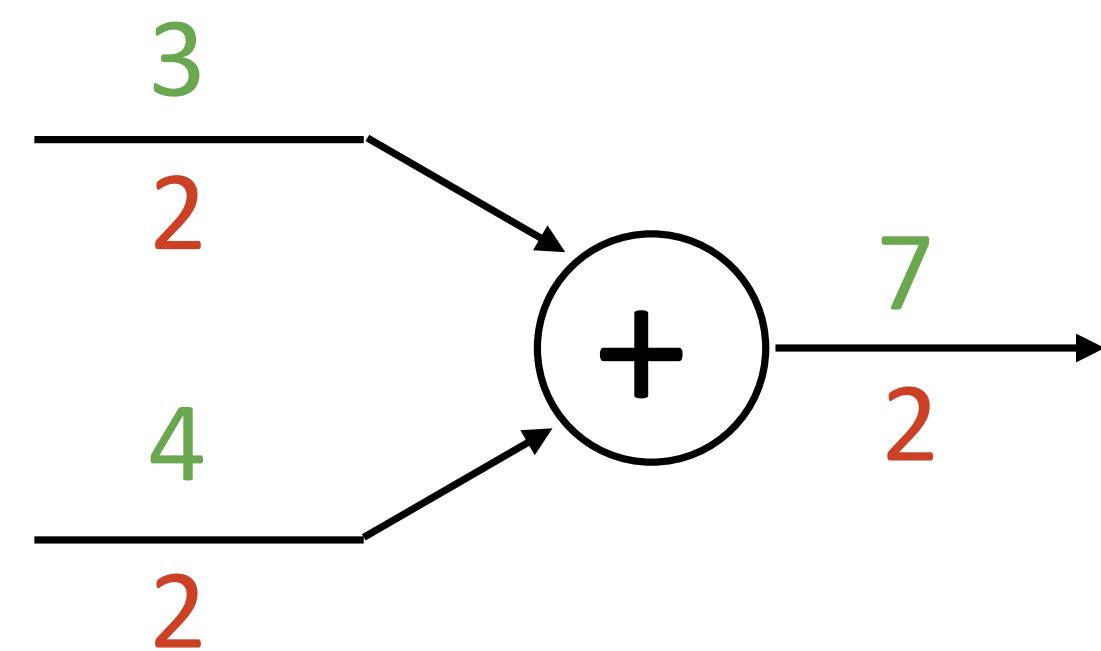


Sigmoid local
gradient:

$$\frac{\partial}{\partial x} [\sigma(x)] = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$

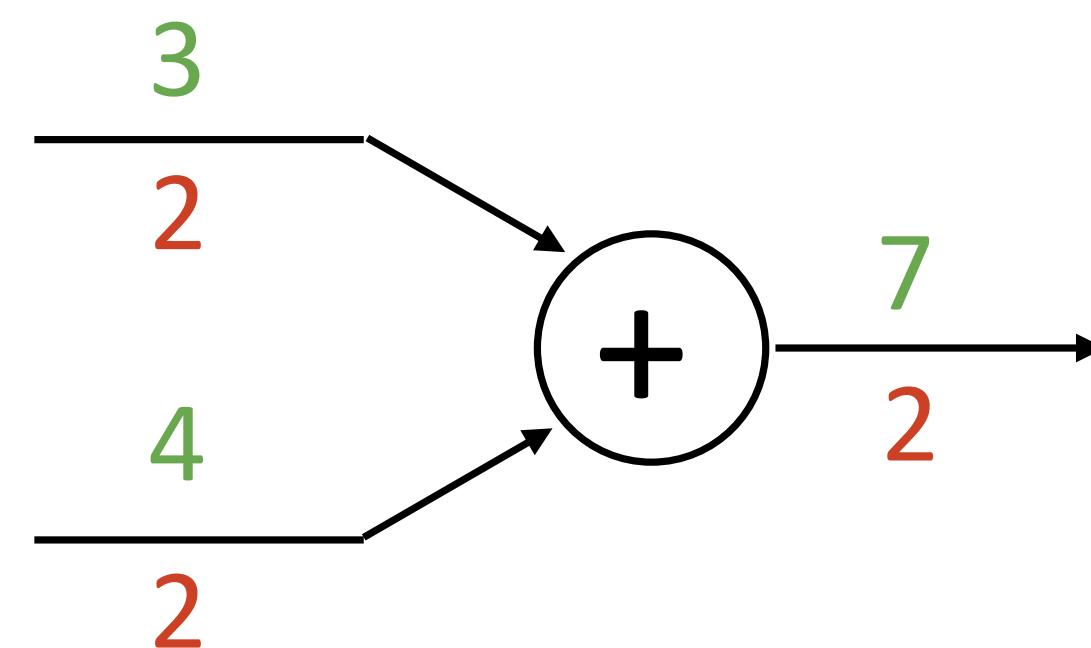
Patterns in Gradient Flow

add gate: gradient distributor

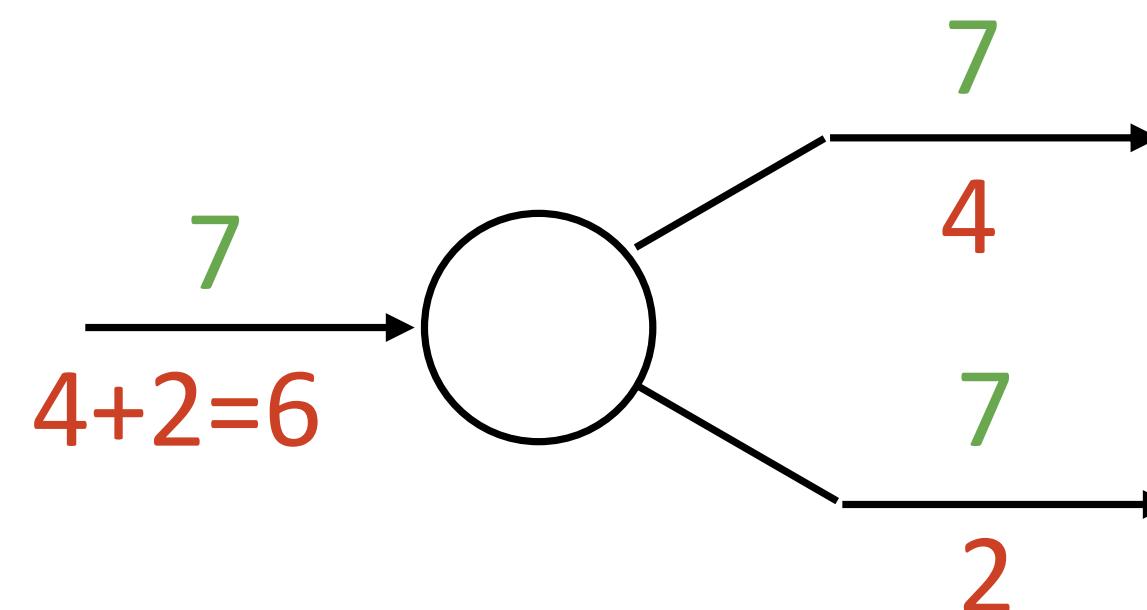


Patterns in Gradient Flow

add gate: gradient distributor

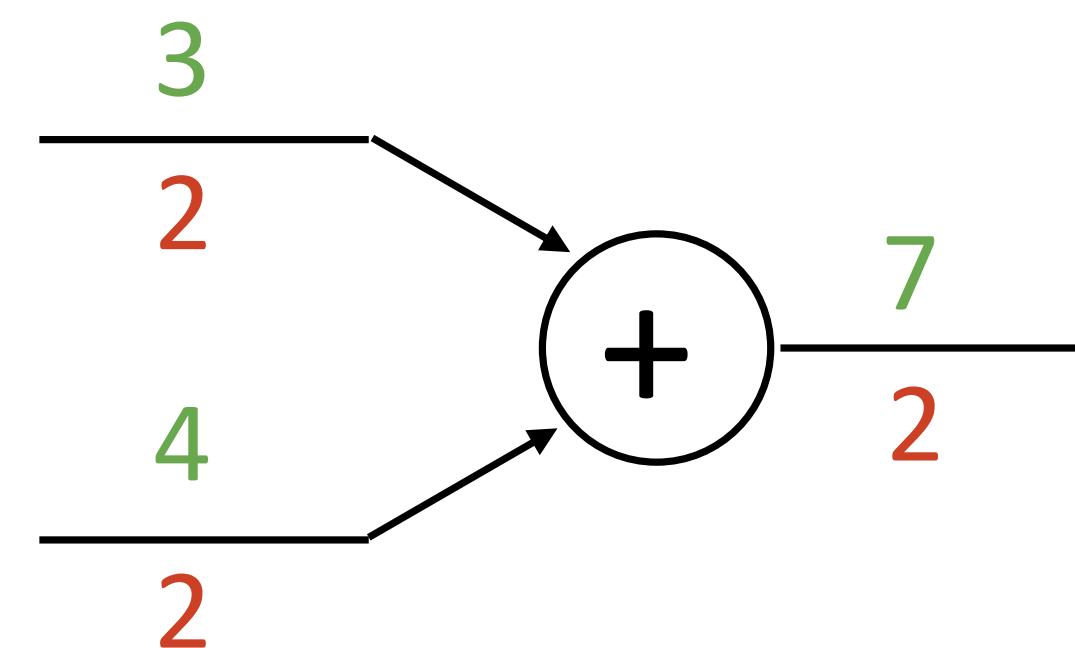


copy gate: gradient adder

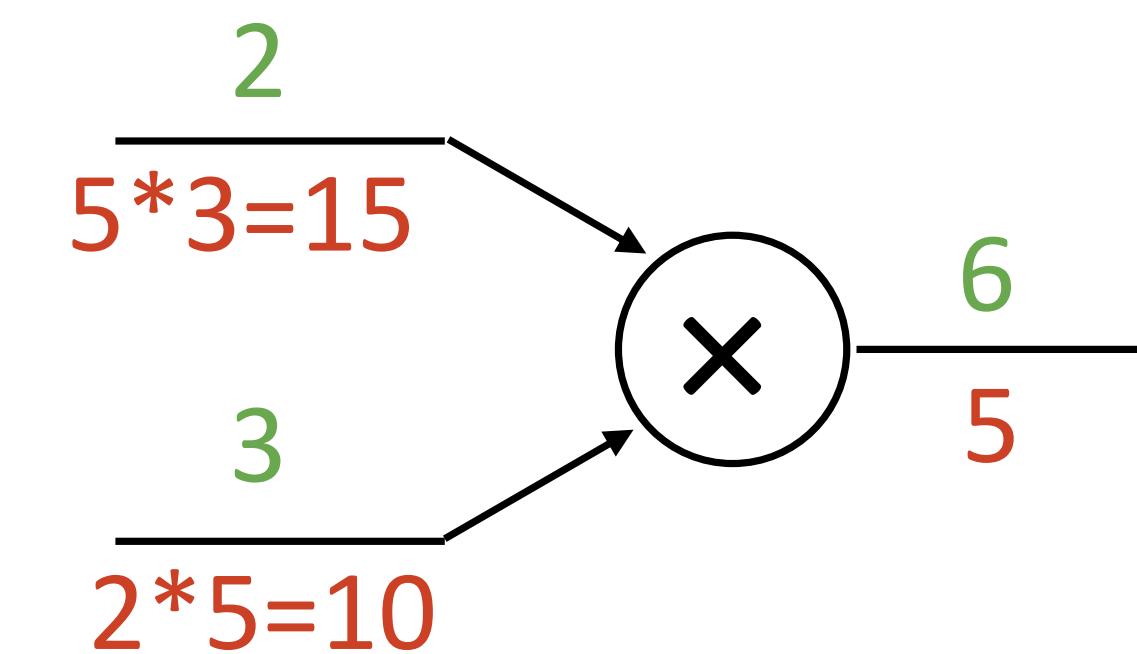


Patterns in Gradient Flow

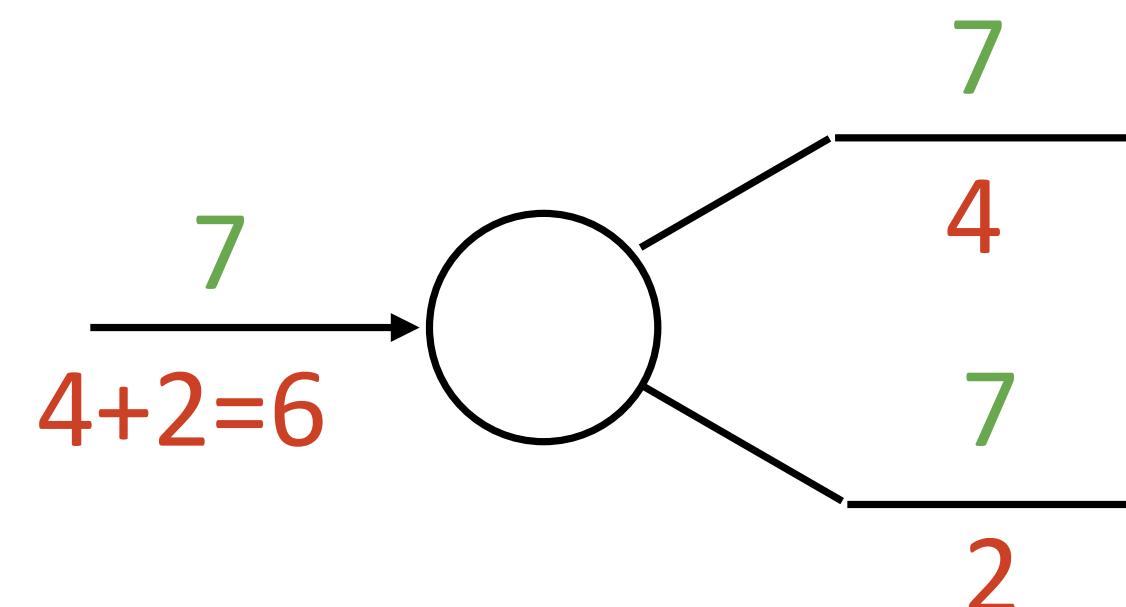
add gate: gradient distributor



mul gate: “swap multiplier”

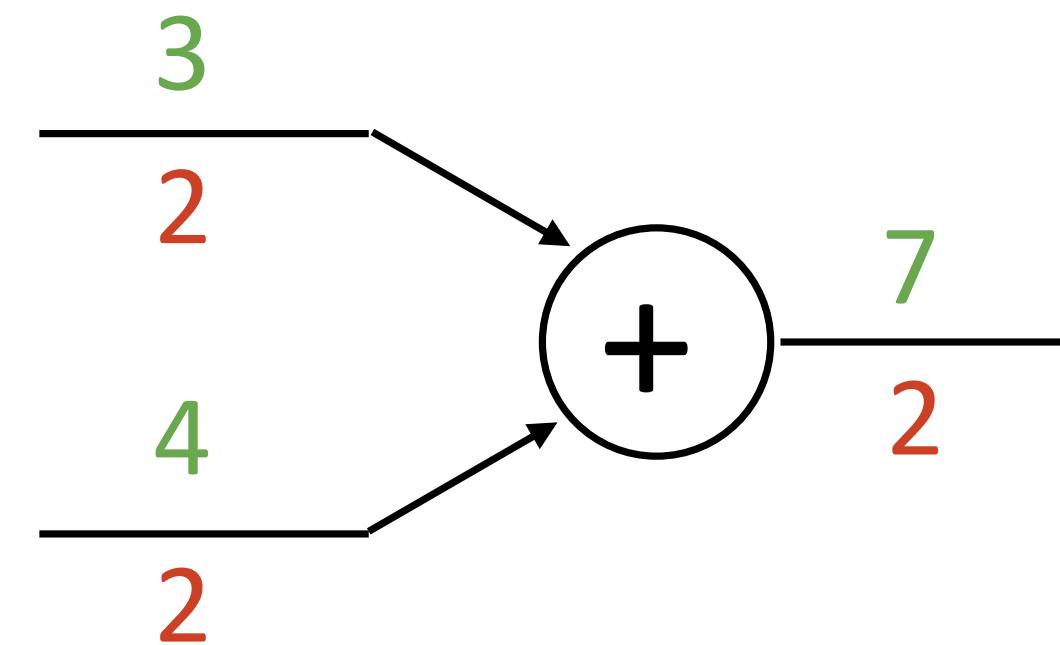


copy gate: gradient adder

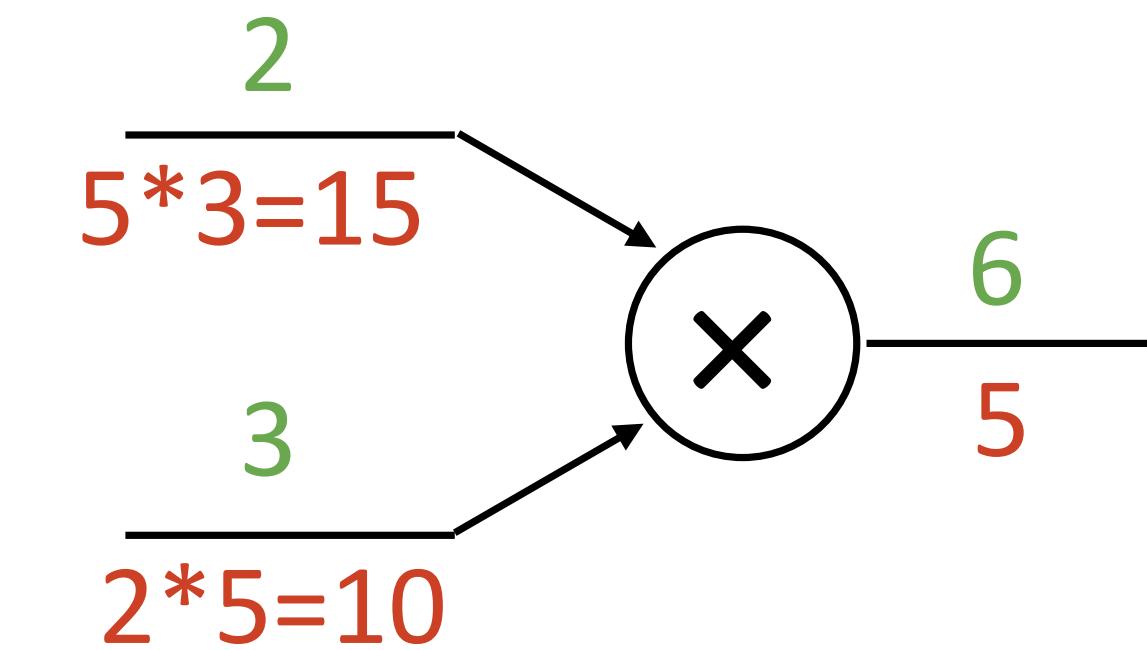


Patterns in Gradient Flow

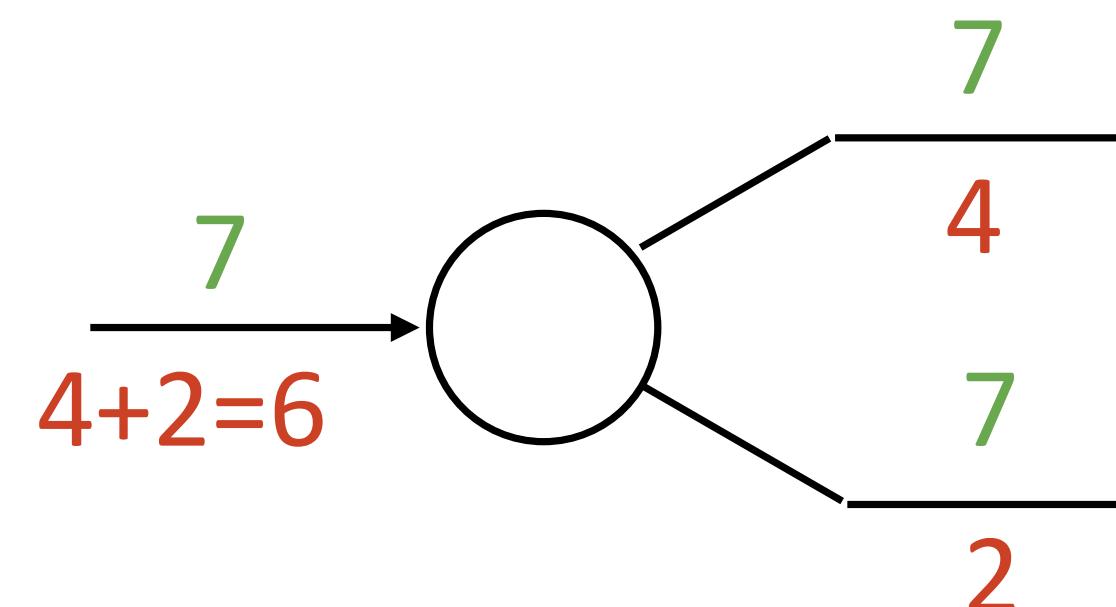
add gate: gradient distributor



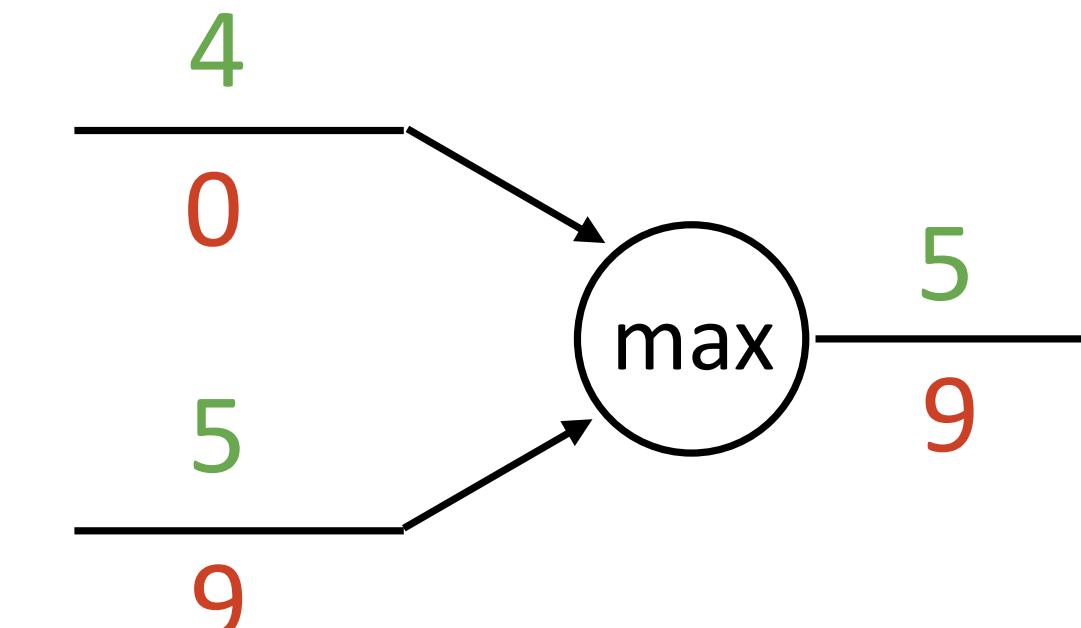
mul gate: “swap multiplier”



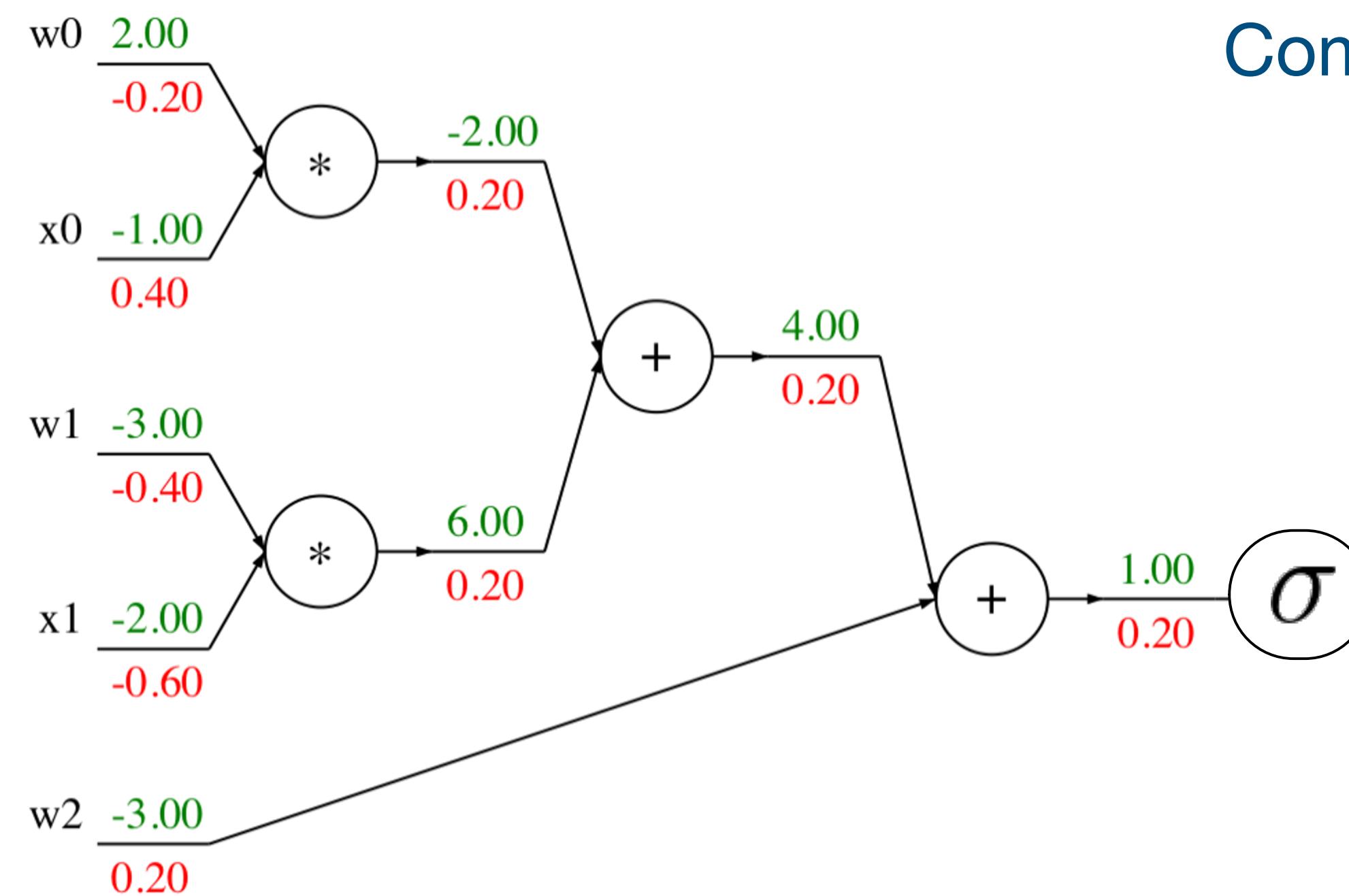
copy gate: gradient adder



max gate: gradient router



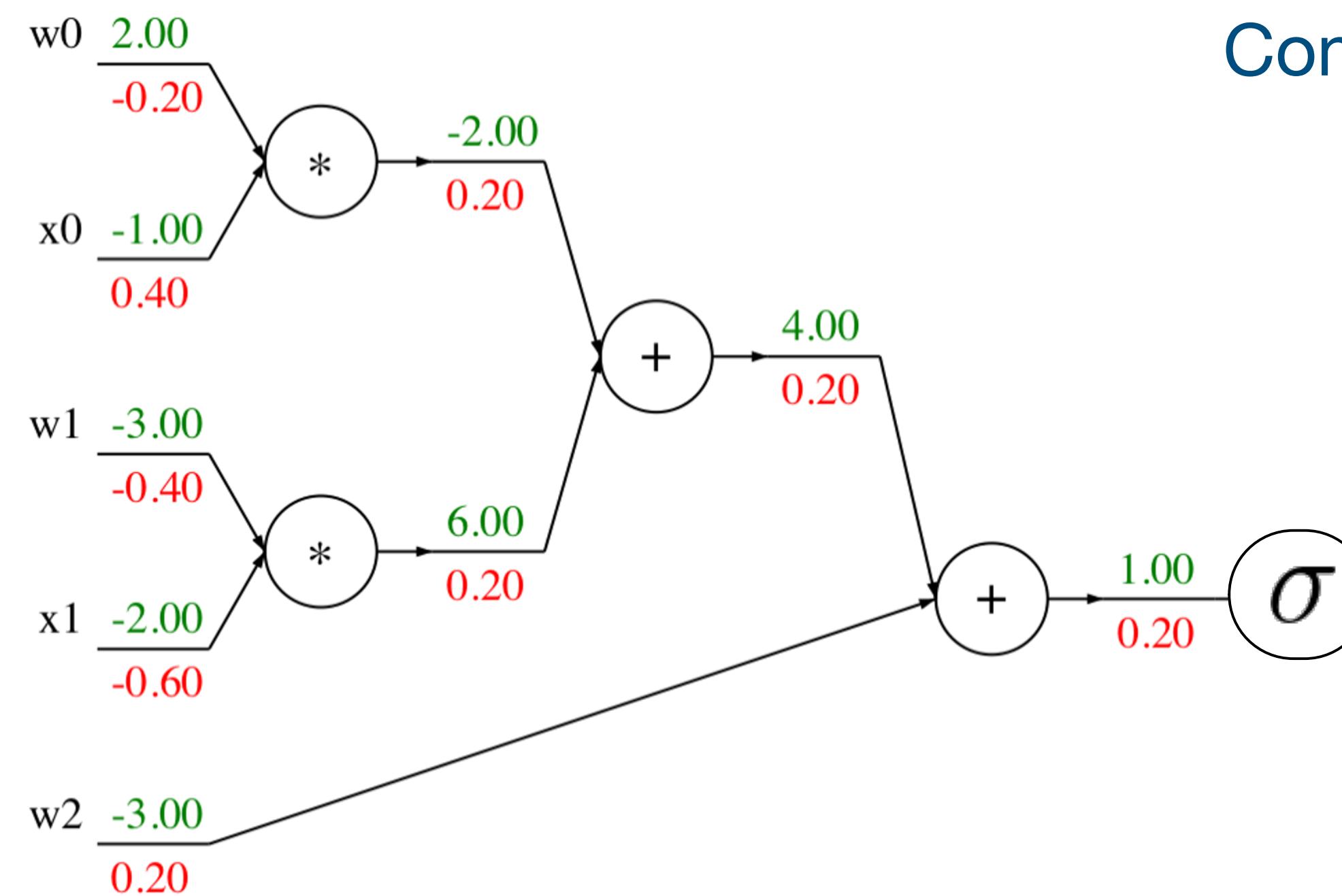
Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):  
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

Backprop Implementation: “Flat” gradient code



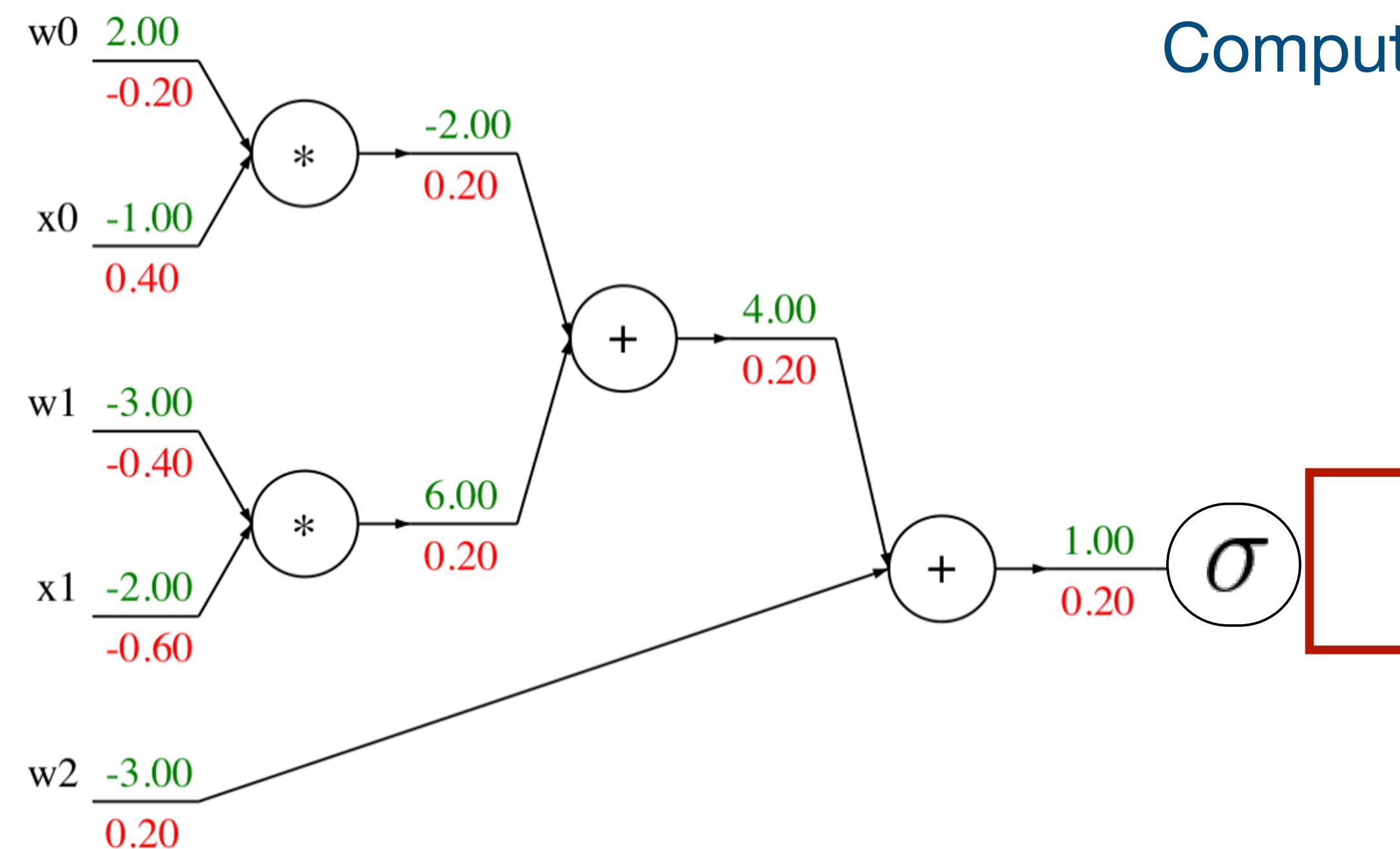
Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute gradients

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

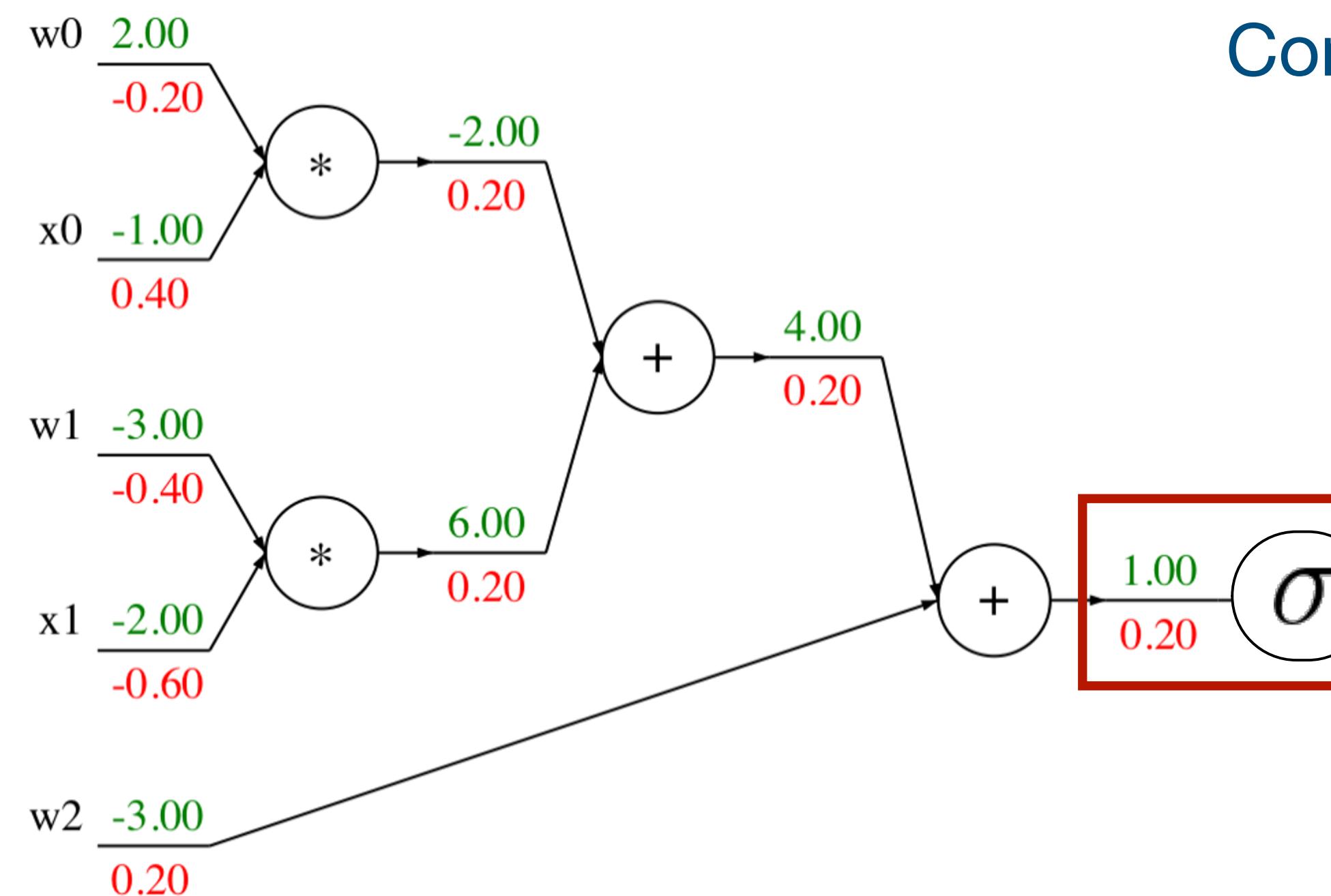
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Base case

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

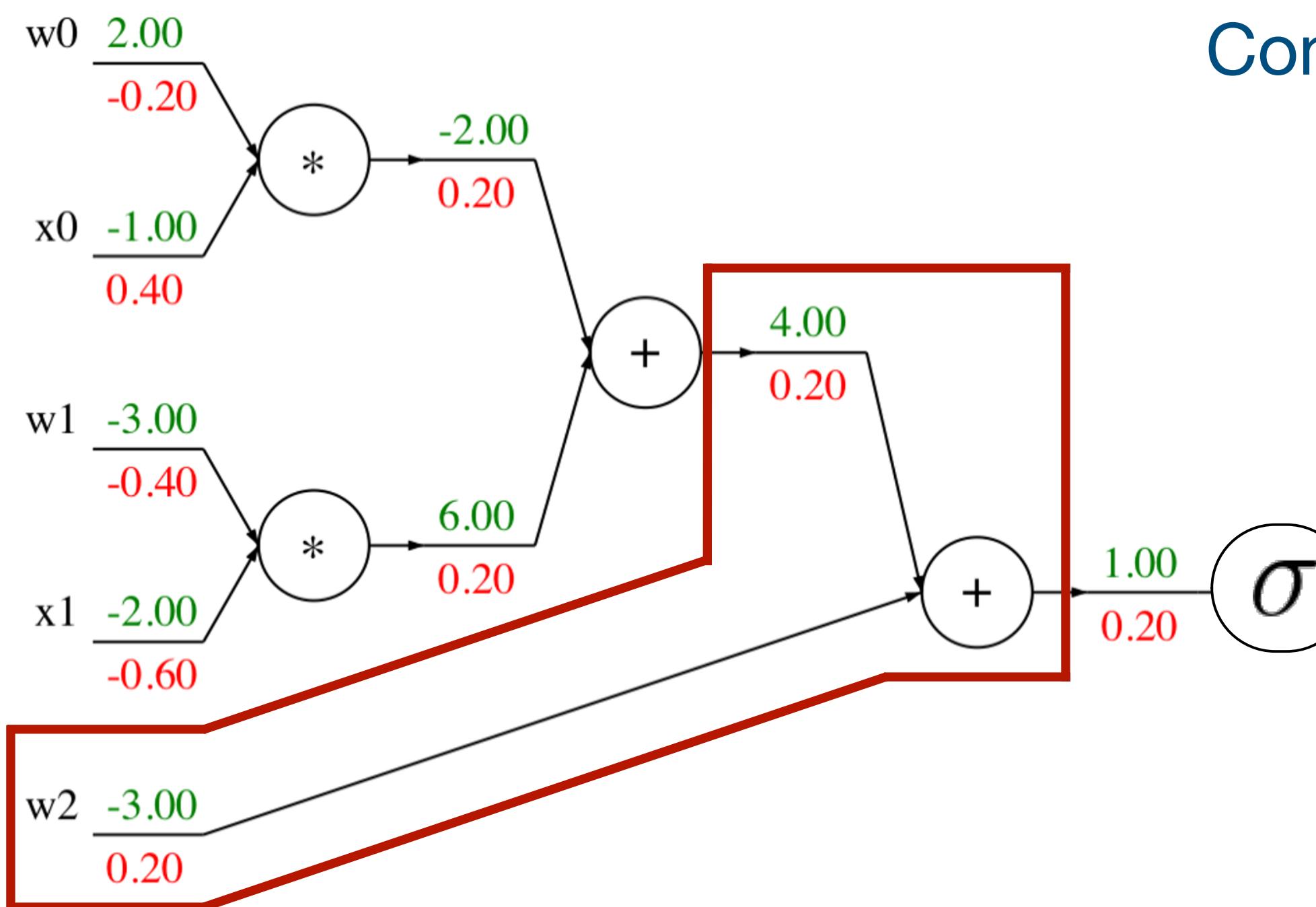
Sigmoid

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

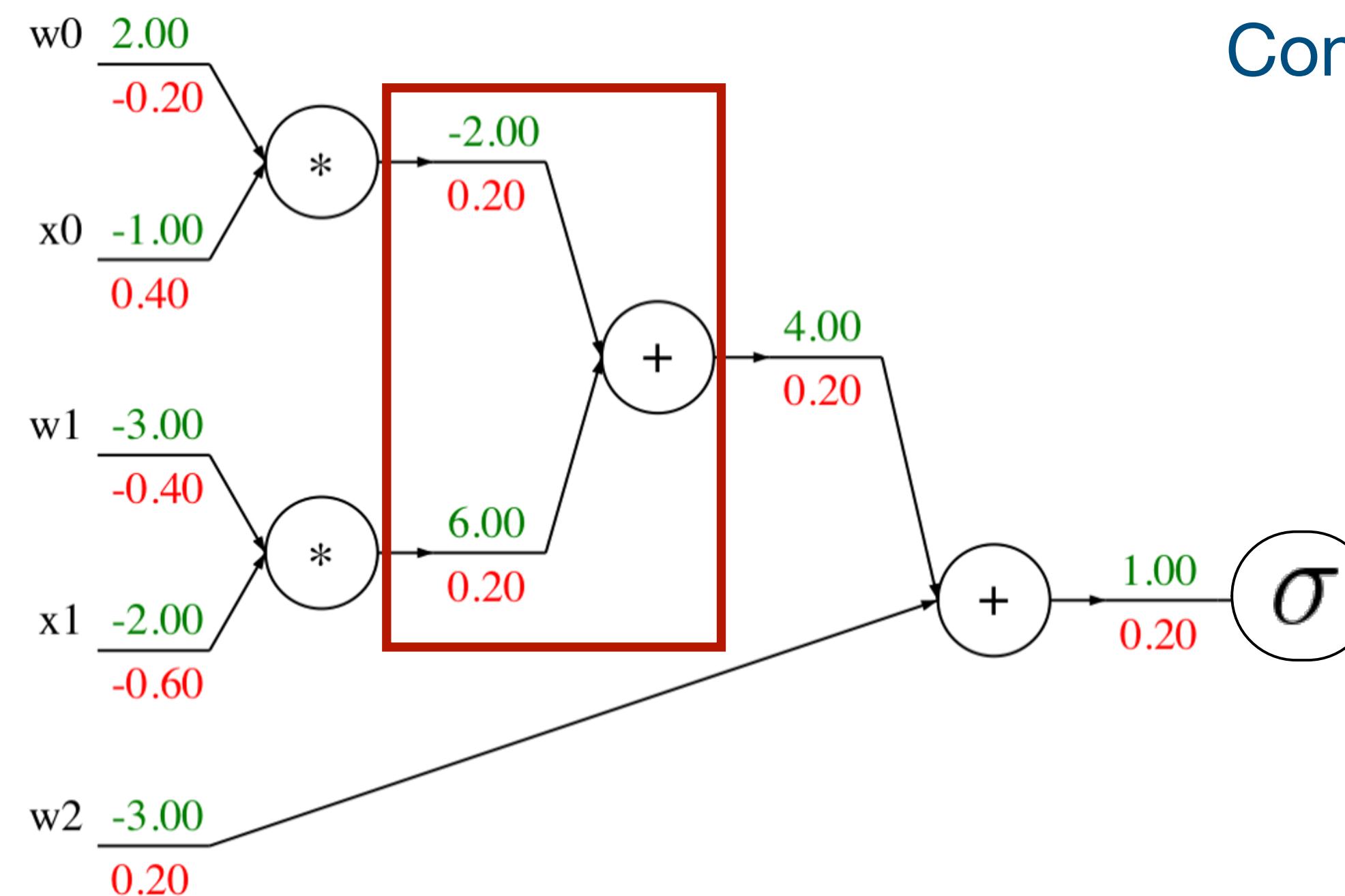
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Add

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

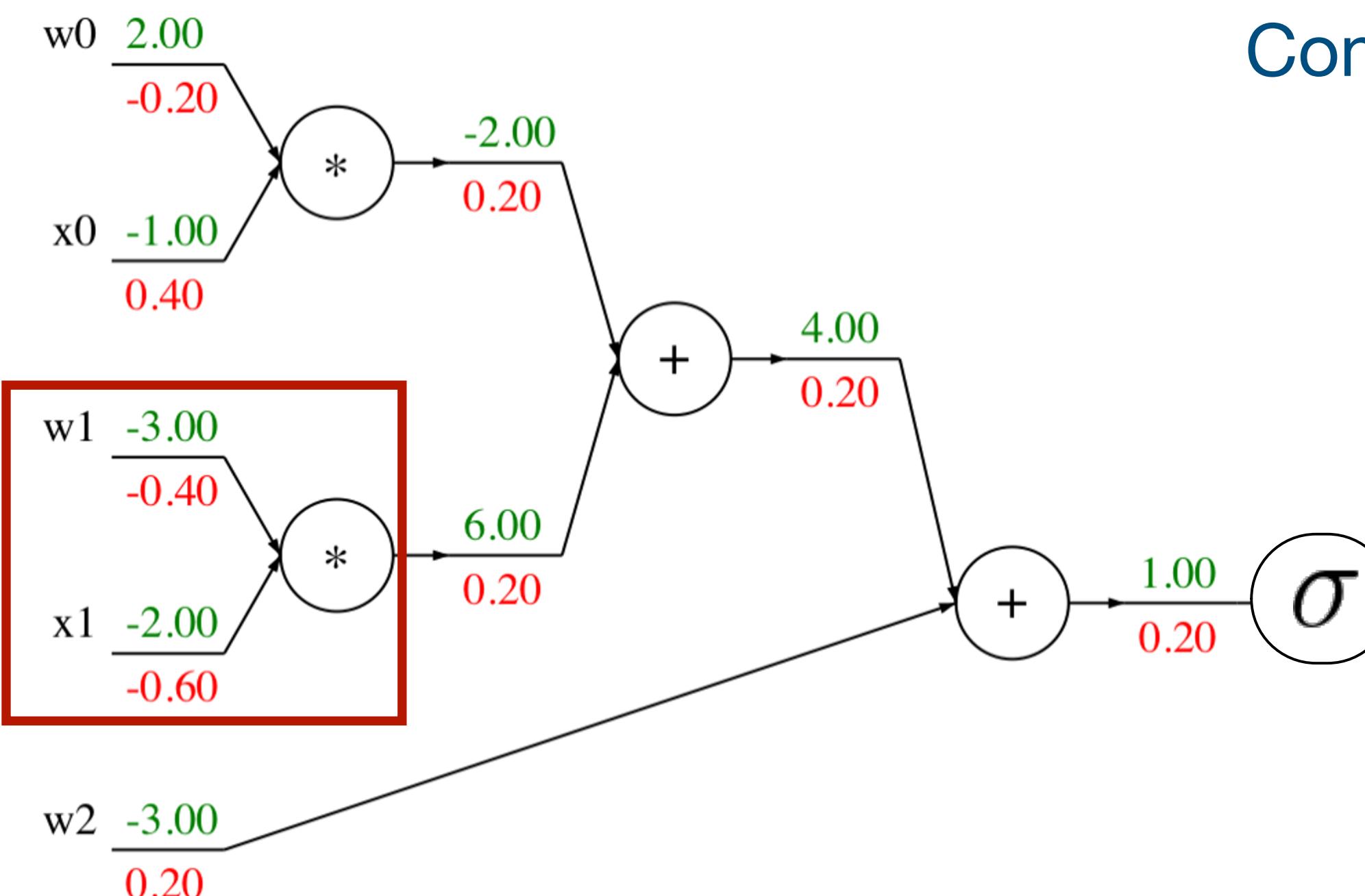
Add

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backward pass:
Compute gradients

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

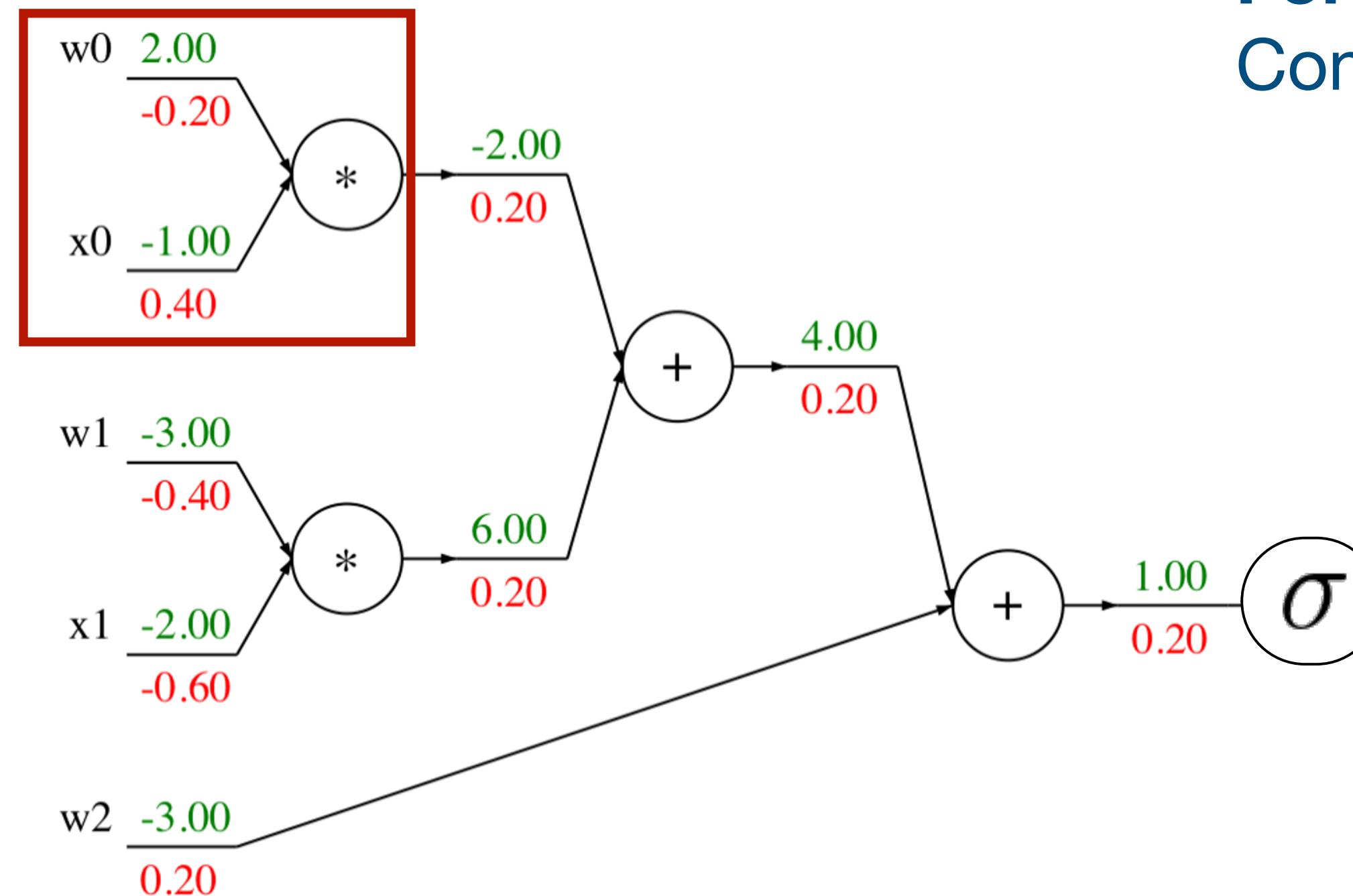
Multiply

Backward pass:
Compute gradients

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backprop Implementation: “Flat” gradient code



Forward pass:
Compute outputs

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Multiply

Backward pass:
Compute gradients

“Flat” Backprop: Do this for Project 1 & 2

Forward pass: Compute outputs

```
#####
# TODO:                                     #
# Implement a vectorized version of the structured SVM loss, storing the   #
# result in loss.                         #
#####
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#####
#                                         END OF YOUR CODE
#####
#
```

Backward pass: Compute gradients

```
#####
# TODO:                                     #
# Implement a vectorized version of the gradient for the structured SVM    #
# loss, storing the result in dW.                                         #
#
# Hint: Instead of computing the gradient from scratch, it may be easier   #
# to reuse some of the intermediate values that you used to compute the    #
# loss.
#####
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#####
#                                         END OF YOUR CODE
#####
#
```



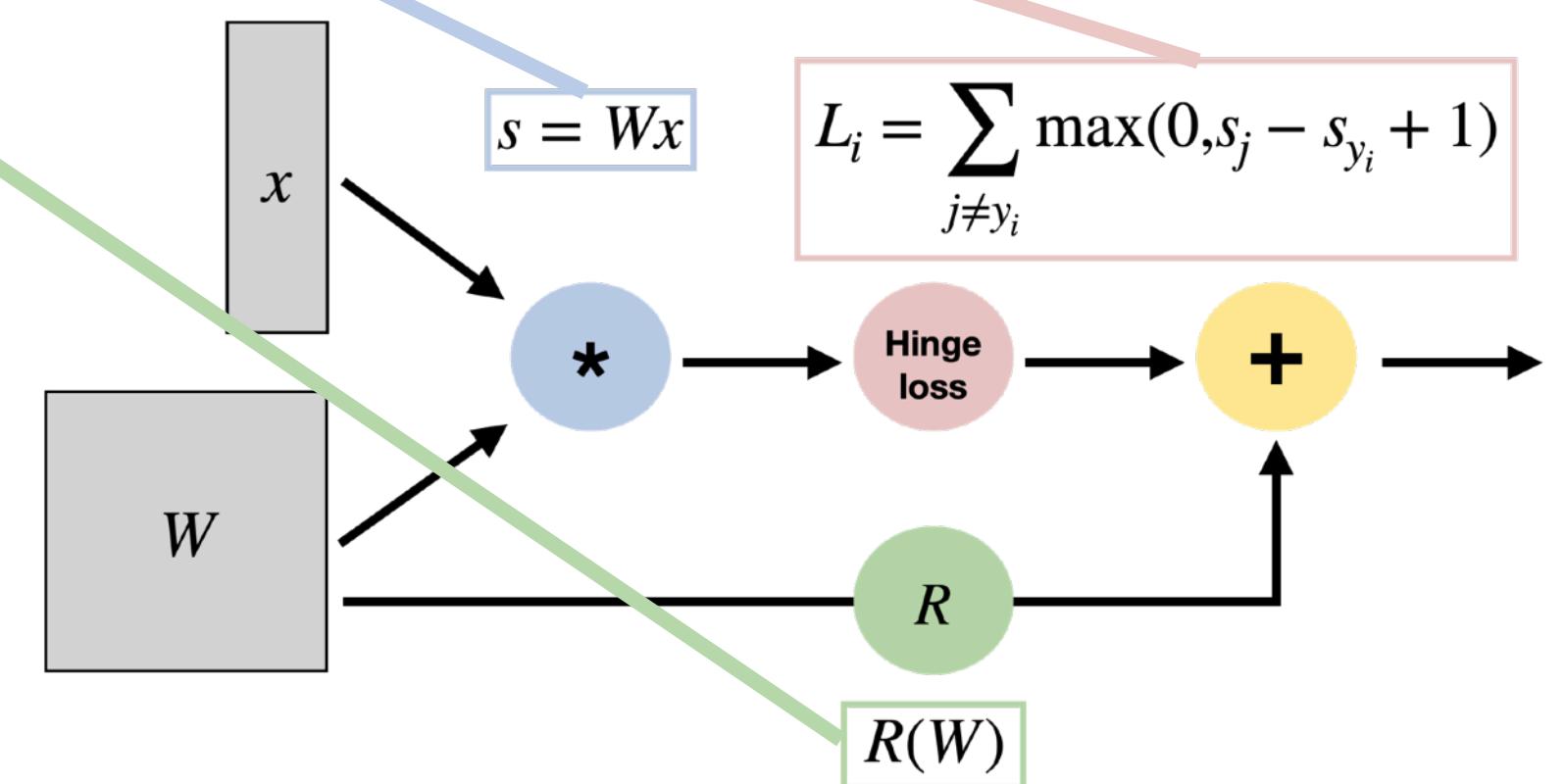
“Flat” Backprop: Do this for Project 1 & 2

Forward pass: Compute outputs

```
#####
# TODO: #
# Implement a vectorized version of the structured SVM loss, storing the #
# result in loss. #
#####
# Replace "pass" statement with your code
num_classes = W.shape[1]
num_train = X.shape[0]
score = # ...
correct_class_score = # ...
margin = # ...
data_loss = # ...
reg_loss = # ...
loss += data_loss + reg_loss
#####
# END OF YOUR CODE
#####
#
```

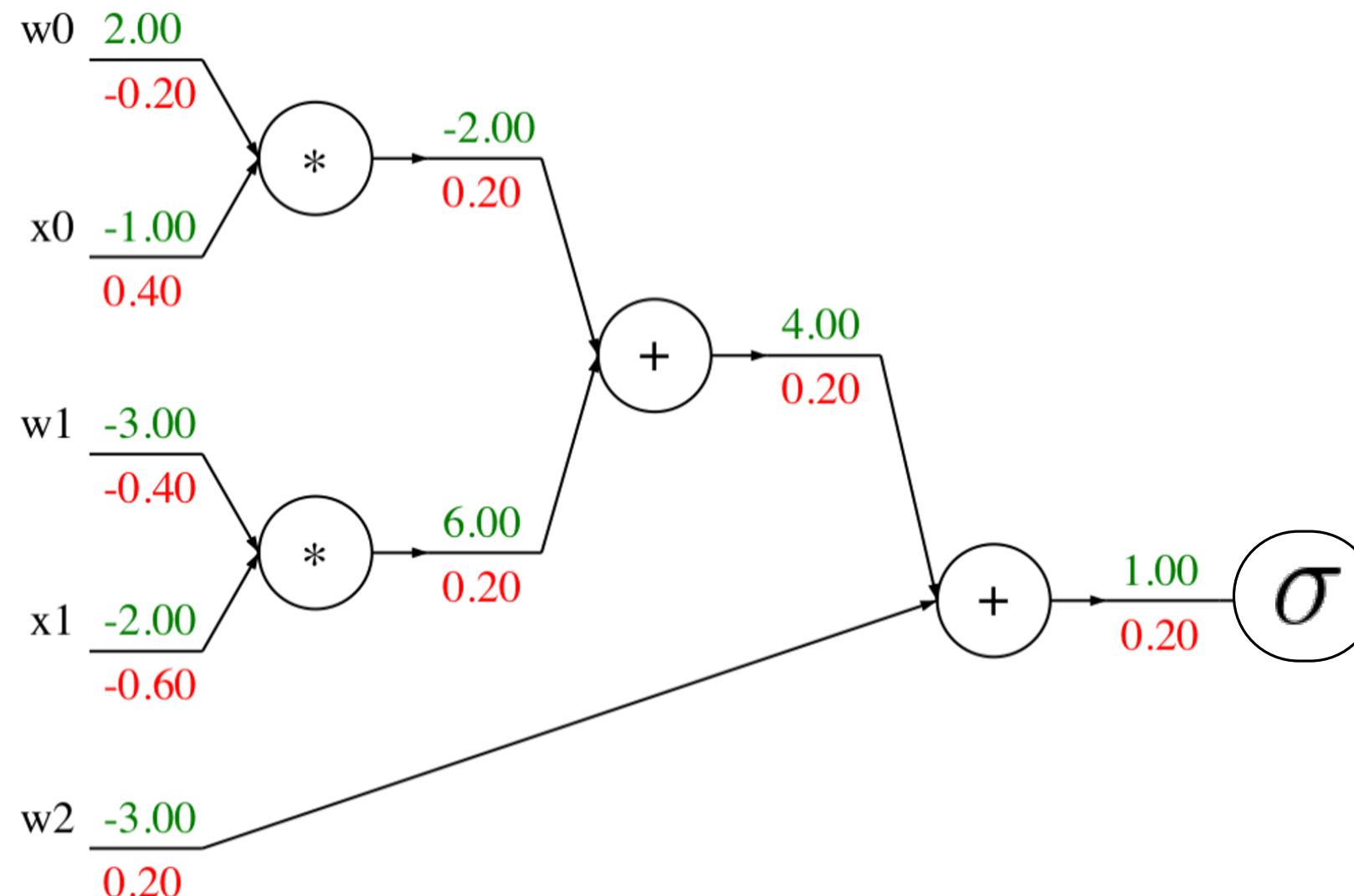
Backward pass: Compute gradients

```
#####
# TODO: #
# Implement a vectorized version of the gradient for the structured SVM #
# loss, storing the result in dW. #
#
# Hint: Instead of computing the gradient from scratch, it may be easier #
# to reuse some of the intermediate values that you used to compute the #
# loss.
#####
# Replace "pass" statement with your code
dmargins = # ...
dscores = # ...
dW = # ...
#####
# END OF YOUR CODE
#####
#
```



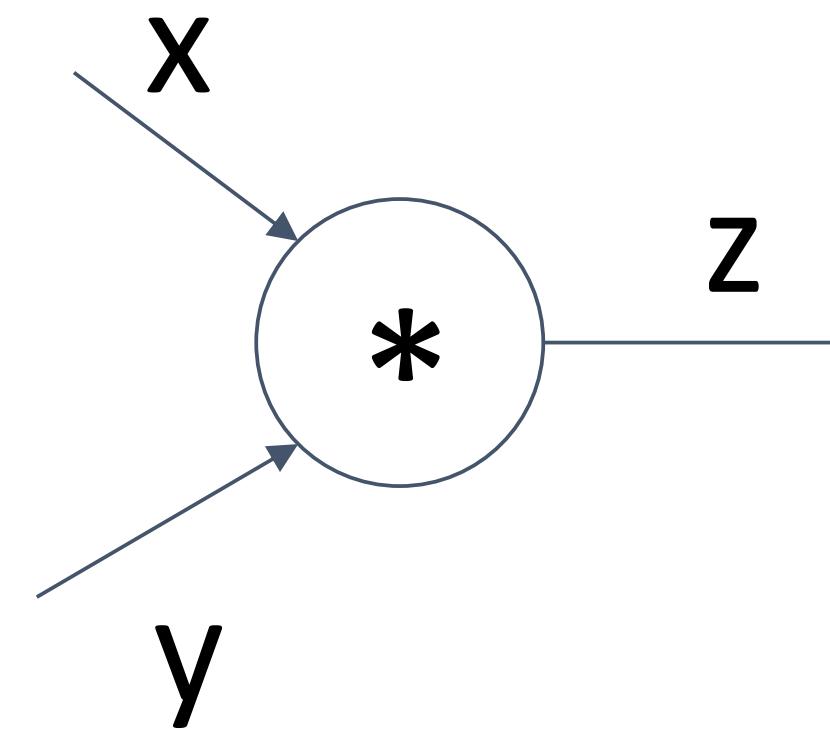
Backprop Implementation: Modular API

Graph (or Net) object (*rough pseudo code*)



```
class ComputationalGraph(object):
    ...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Example: PyTorch Autograd Functions



(x, y, z are scalars)

```
class Multiply(torch.autograd.Function):  
    @staticmethod  
    def forward(ctx, x, y):  
        ctx.save_for_backward(x, y)  
        z = x * y  
        return z  
  
    @staticmethod  
    def backward(ctx, grad_z):  
        x, y = ctx.saved_tensors  
        grad_x = y * grad_z #  $dz/dx * dL/dz$   
        grad_y = x * grad_z #  $dz/dy * dL/dz$   
        return grad_x, grad_y
```

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

So far: backprop with scalars

What about vector-valued functions?

Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^N, \\ \left(\frac{\partial y}{\partial x}\right)_i &= \frac{\partial y}{\partial x_i}\end{aligned}$$

For each element of x , if it changes by a small amount then how much will y change?



Recap: Vector Derivatives

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^N, \\ \left(\frac{\partial y}{\partial x}\right)_i &= \frac{\partial y}{\partial x_i}\end{aligned}$$

For each element of x , if it changes by a small amount then how much will y change?

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

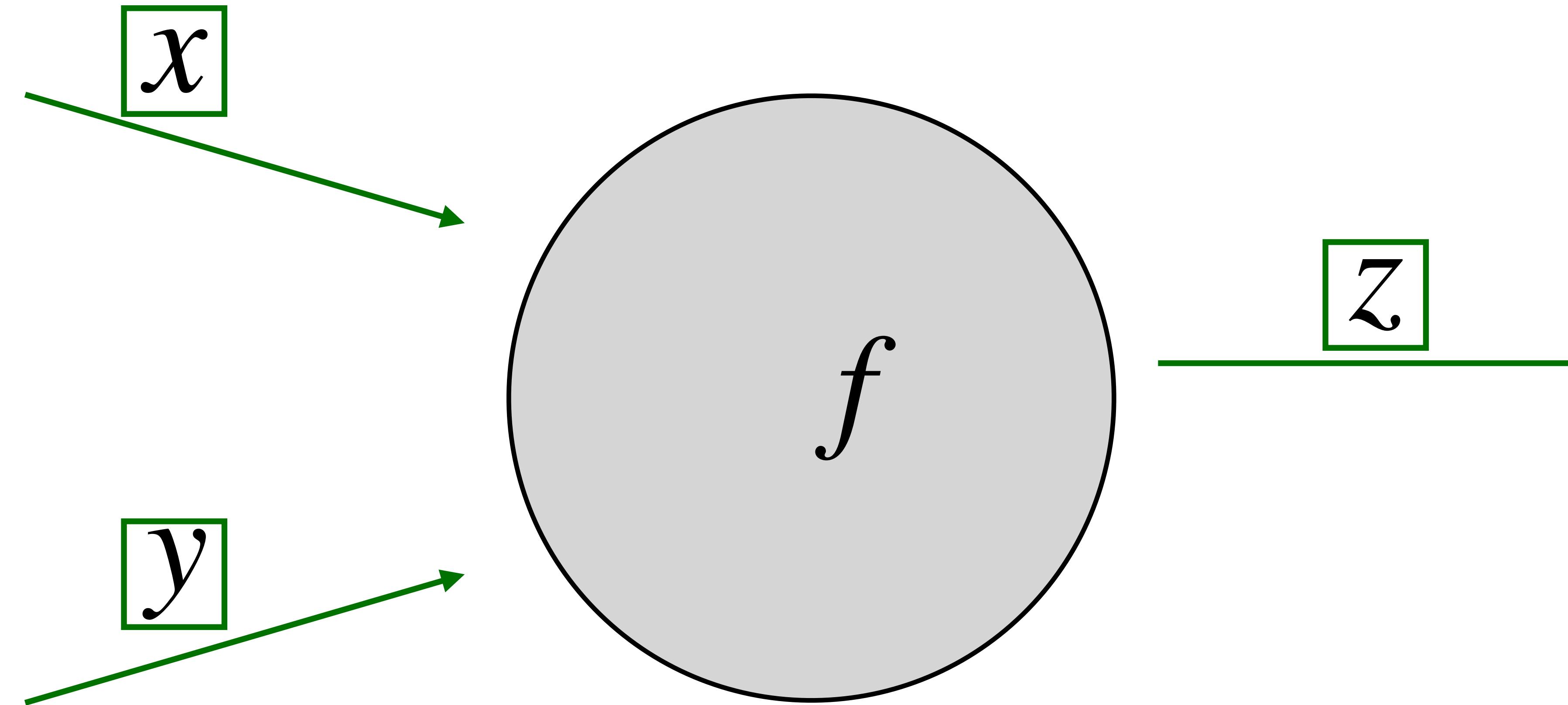
Derivative is **Jacobian**:

$$\begin{aligned}\frac{\partial y}{\partial x} &\in \mathbb{R}^{N \times M} \\ \left(\frac{\partial y}{\partial x}\right)_{i,j} &= \frac{\partial y_j}{\partial x_i}\end{aligned}$$

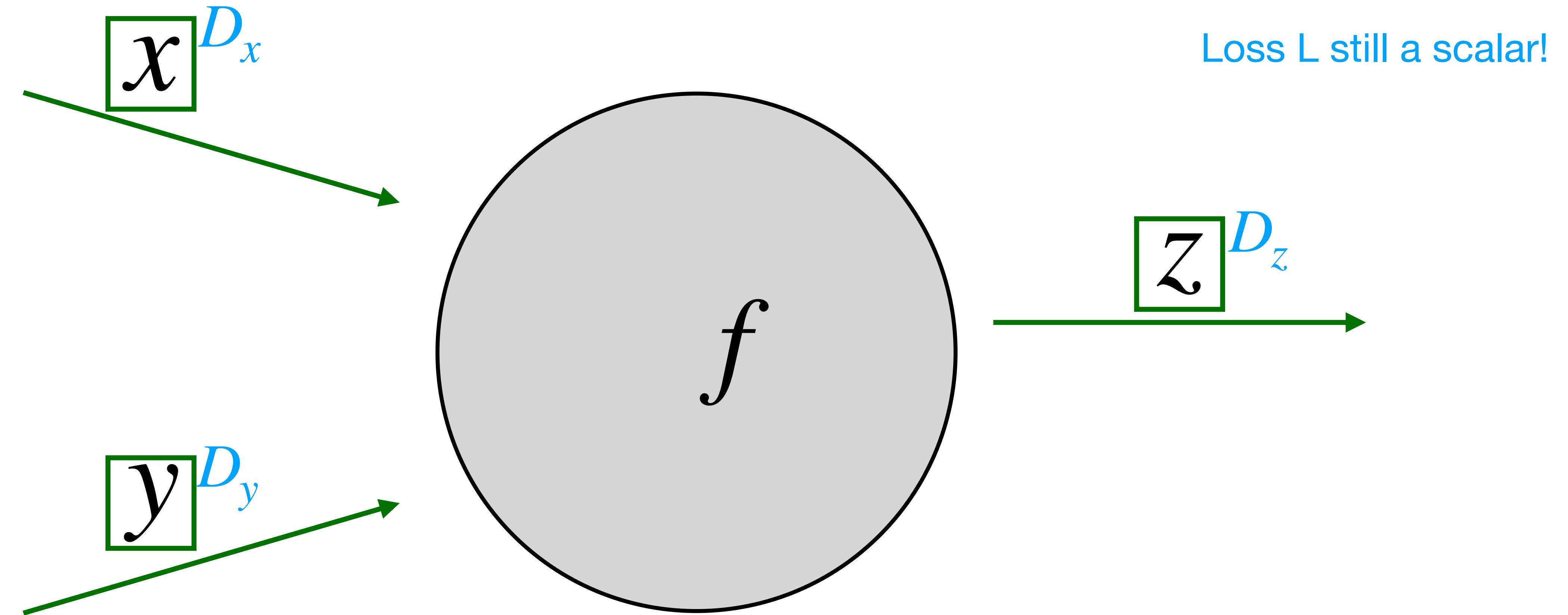
For each element of x , if it changes by a small amount then how much will each element of y change?



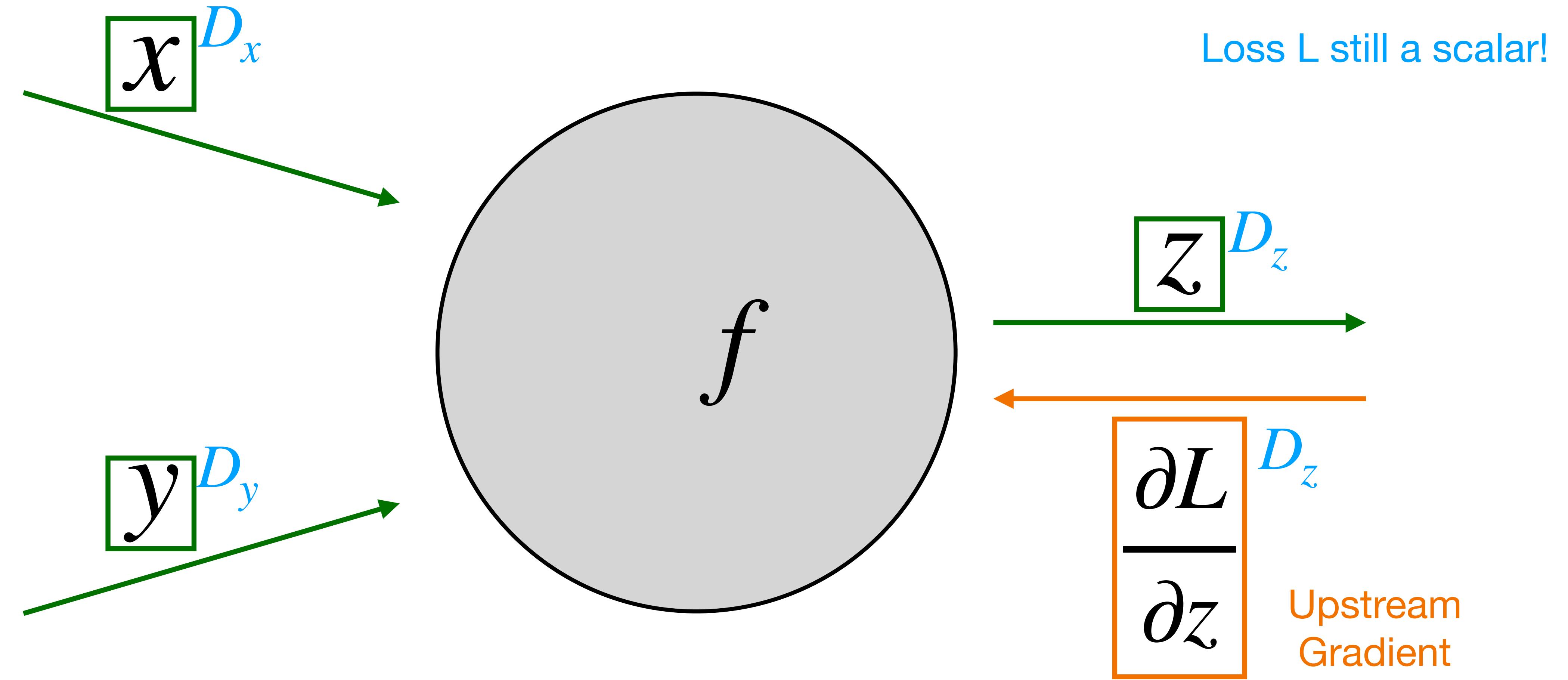
Backprop with Vectors



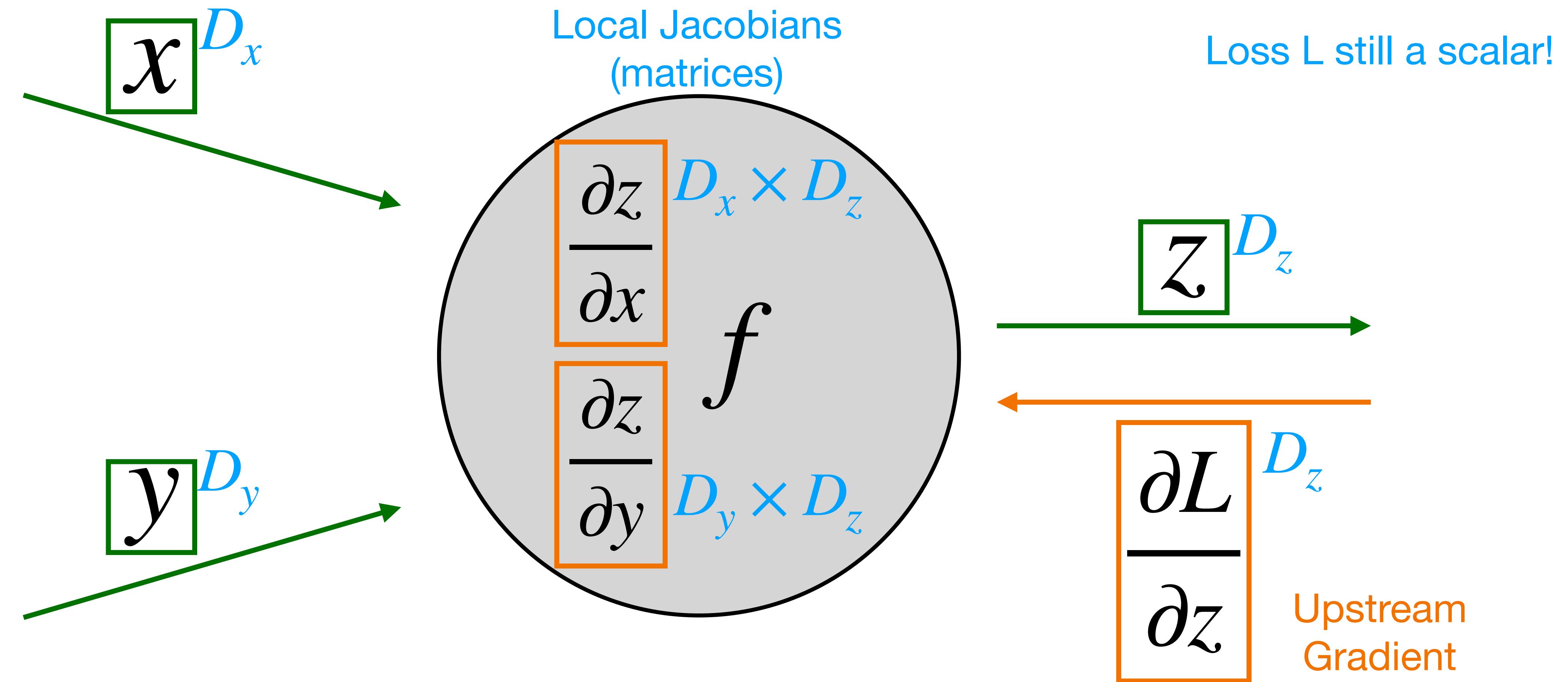
Backprop with Vectors



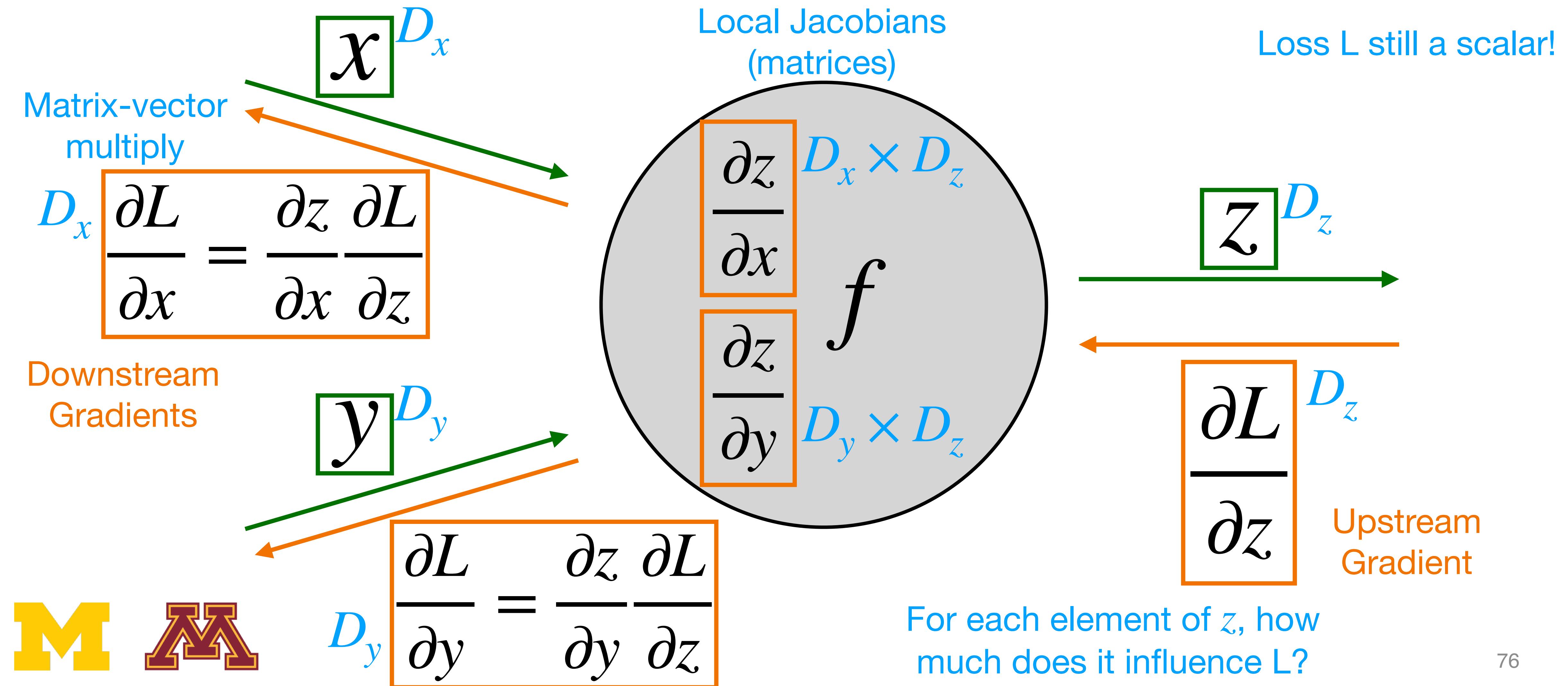
Backprop with Vectors



Backprop with Vectors



Backprop with Vectors



Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$

$$f(x) = \max(0, x)$$

(elementwise)

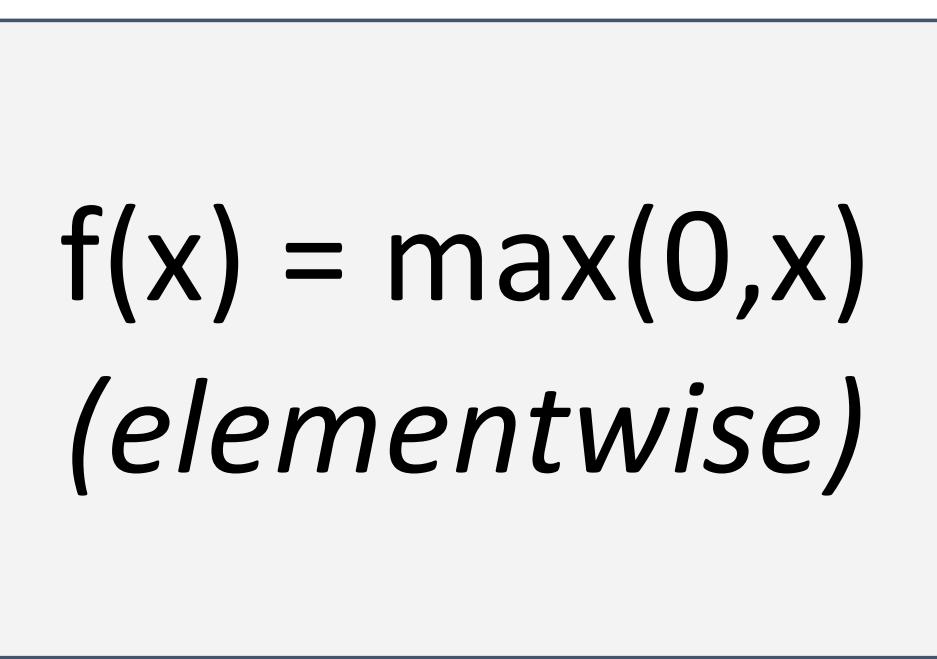
4D output y:

$$\begin{array}{l} \longrightarrow [1] \\ \longrightarrow [0] \\ \longrightarrow [3] \\ \longrightarrow [0] \end{array}$$

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D dL/dy :

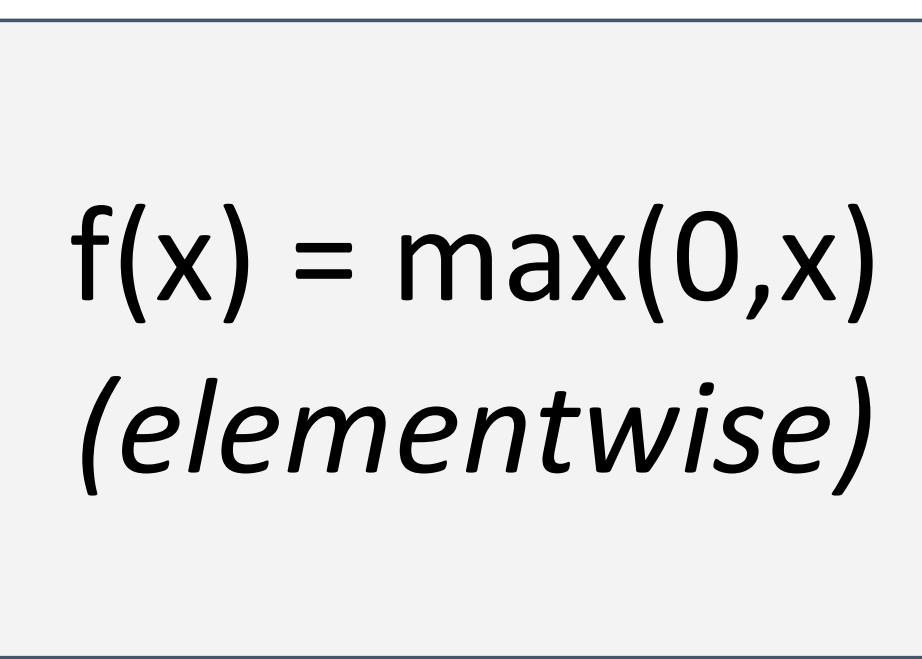
$$\begin{array}{c} \longleftarrow [4] \longrightarrow \\ \longleftarrow [-1] \longrightarrow \\ \longleftarrow [5] \longrightarrow \\ \longleftarrow [9] \longrightarrow \end{array}$$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$[dy/dx] [dL/dy]$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

4D dL/dy :

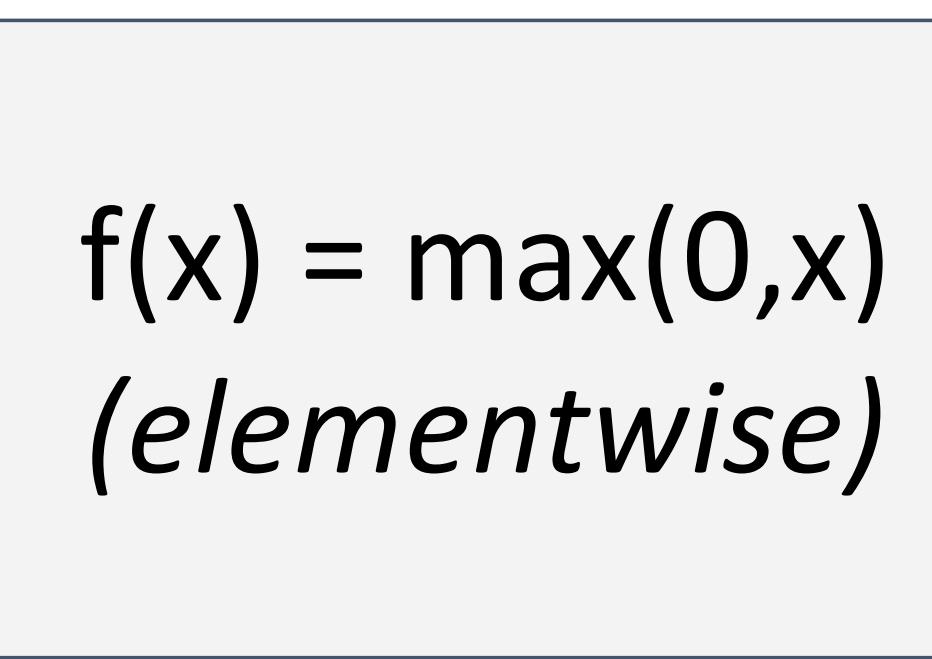
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$



4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

4D dL/dx :

$[4]$	\longleftarrow	$[1 \ 0 \ 0 \ 0]$	$[4]$
$[0]$	\longleftarrow	$[0 \ 0 \ 0 \ 0]$	$[-1]$
$[5]$	\longleftarrow	$[0 \ 0 \ 1 \ 0]$	$[5]$
$[0]$	\longleftarrow	$[0 \ 0 \ 0 \ 0]$	$[9]$

4D dL/dy :

$[4]$	\longleftarrow	$[4]$
$[-1]$	\longleftarrow	$[-1]$
$[5]$	\longleftarrow	$[5]$
$[9]$	\longleftarrow	$[9]$

Upstream
gradient

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \xrightarrow{\text{green arrow}}$$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output y:

$$\xrightarrow{\text{green arrow}} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Jacobian is **sparse**: off-diagonal entries all zero!
Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx :

$$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix} \xleftarrow{\text{red arrow}} \begin{bmatrix} dy/dx \\ dL/dy \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix}$$

4D dL/dy :

$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \xleftarrow{\text{red arrow}} \begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \end{array} \quad \begin{array}{l} \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \\ \xleftarrow{\text{red arrow}} \end{array} \quad \begin{array}{l} \text{Upstream} \\ \text{gradient} \end{array}$$

Backprop with Vectors

4D input x:

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \longrightarrow$$

$$f(x) = \max(0, x) \\ (\textit{elementwise})$$

4D output y:

$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Jacobian is **sparse**: off-diagonal entries all zero!
Never **explicitly** form Jacobian; instead use **implicit** multiplication

4D dL/dx :

$$\begin{bmatrix} 4 \\ 0 \\ 5 \\ 0 \end{bmatrix} \leftarrow$$

$[dy/dx] [dL/dy]$

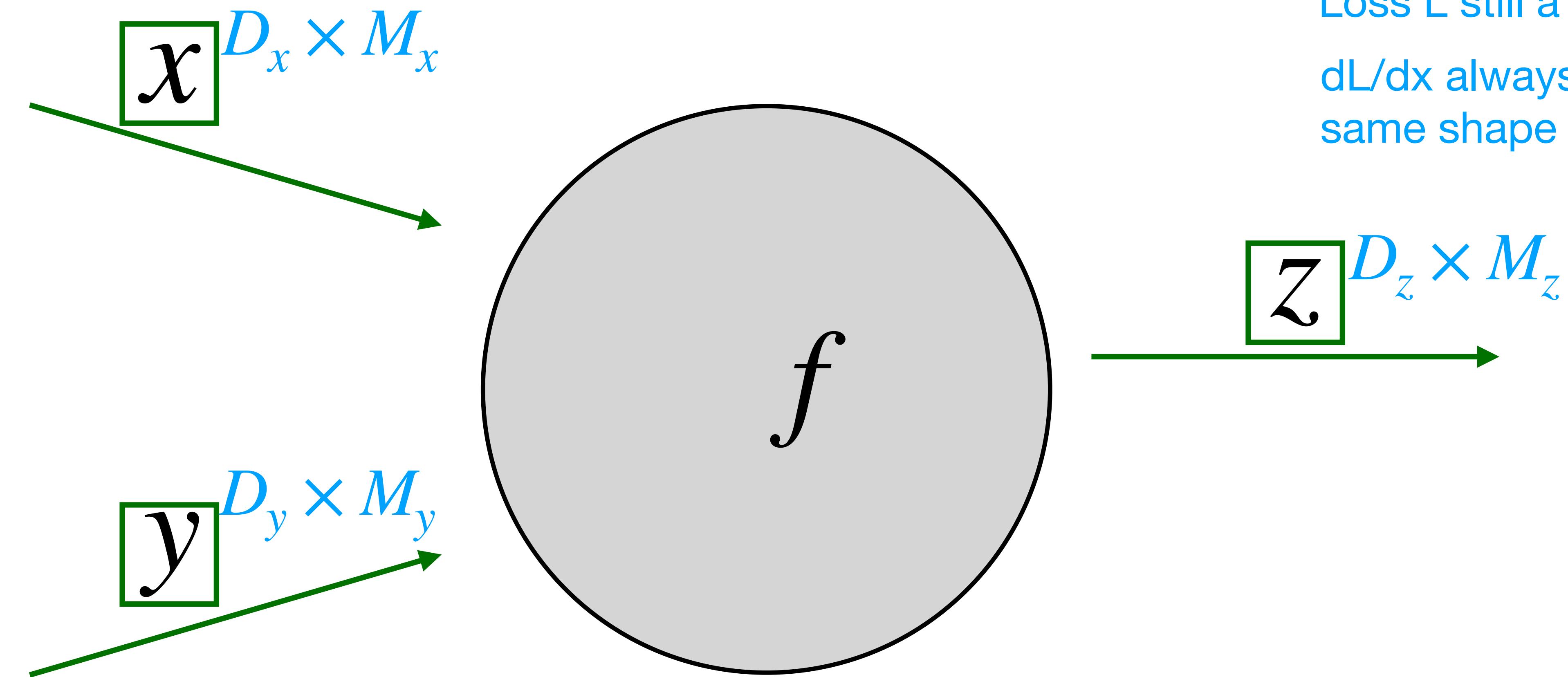
$$\left(\frac{\partial L}{\partial x} \right)_i = \begin{cases} \left(\frac{\partial L}{\partial y} \right)_i, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

4D dL/dy :

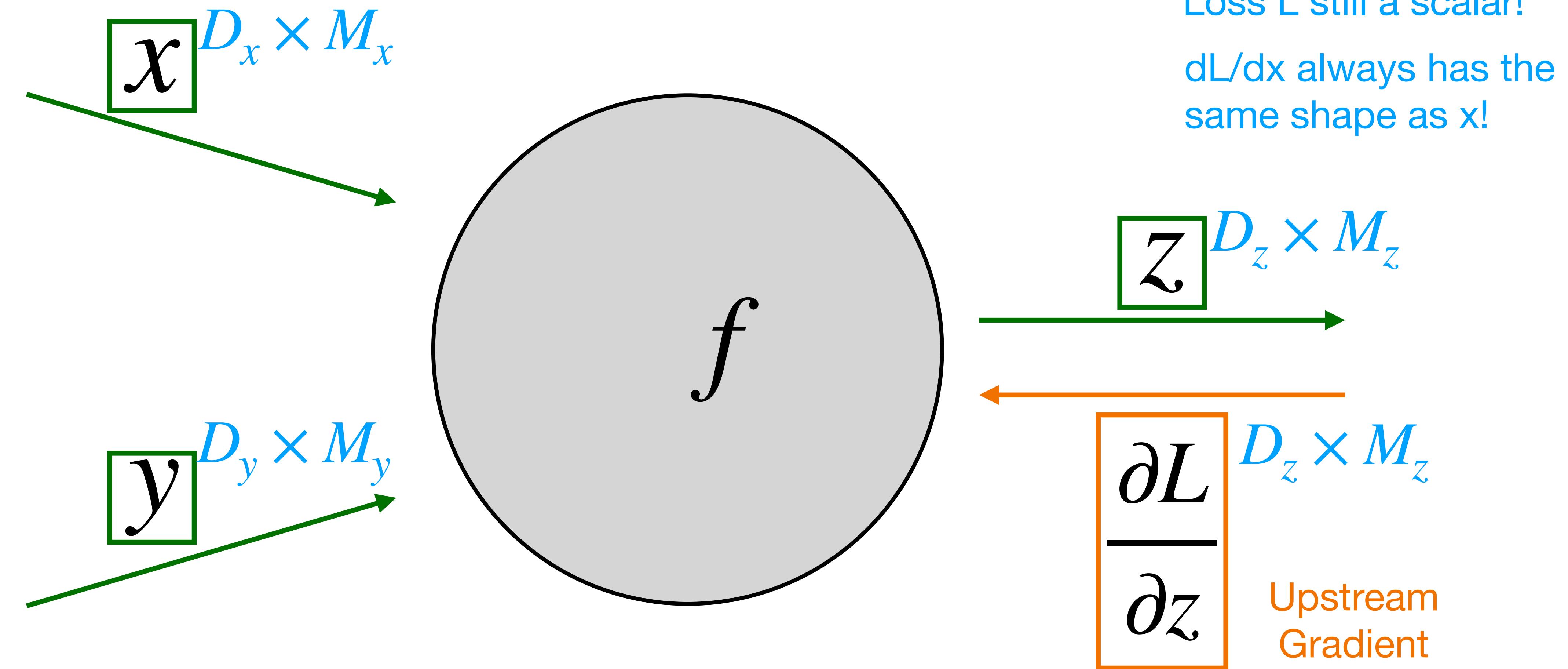
$$\begin{bmatrix} 4 \\ -1 \\ 5 \\ 9 \end{bmatrix} \leftarrow \begin{array}{c} \leftarrow \\ \leftarrow \\ \leftarrow \\ \leftarrow \end{array}$$

Upstream
gradient

Backprop with Matrices (or Tensors)

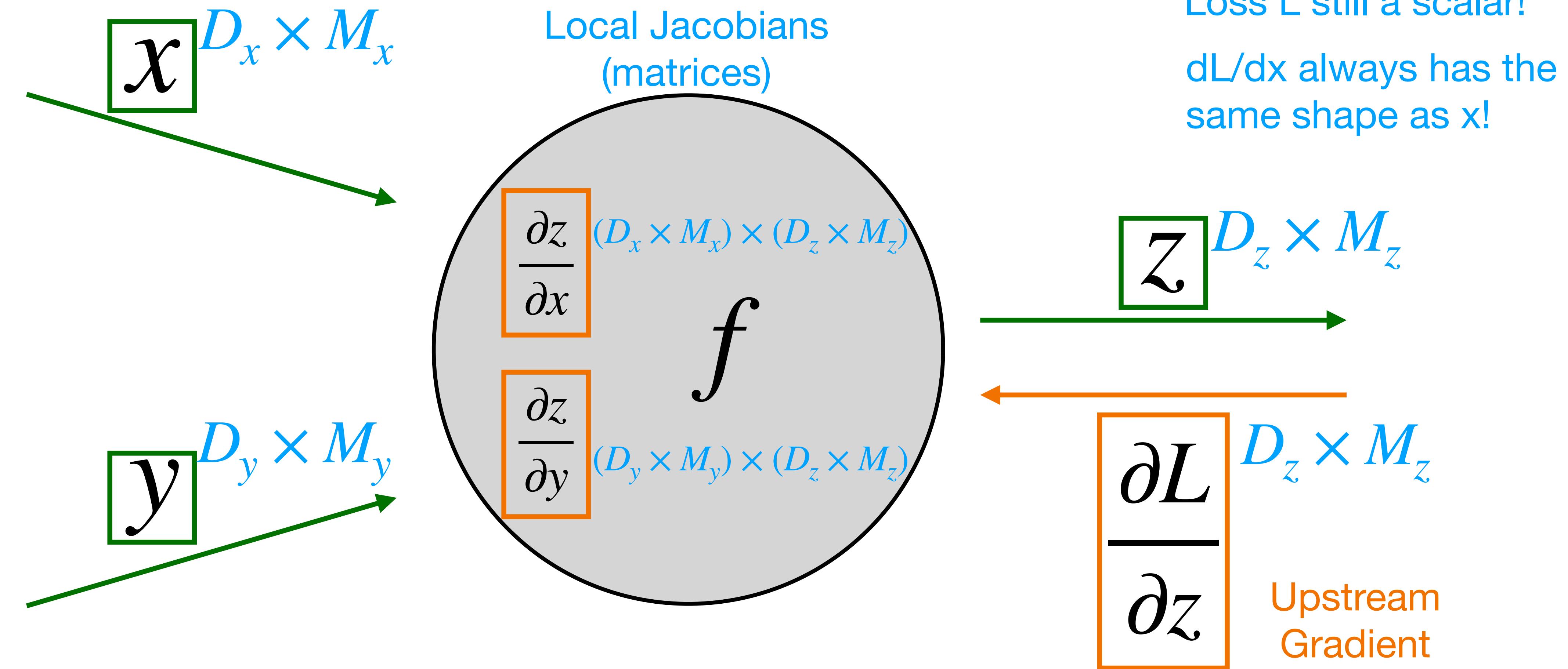


Backprop with Matrices (or Tensors)



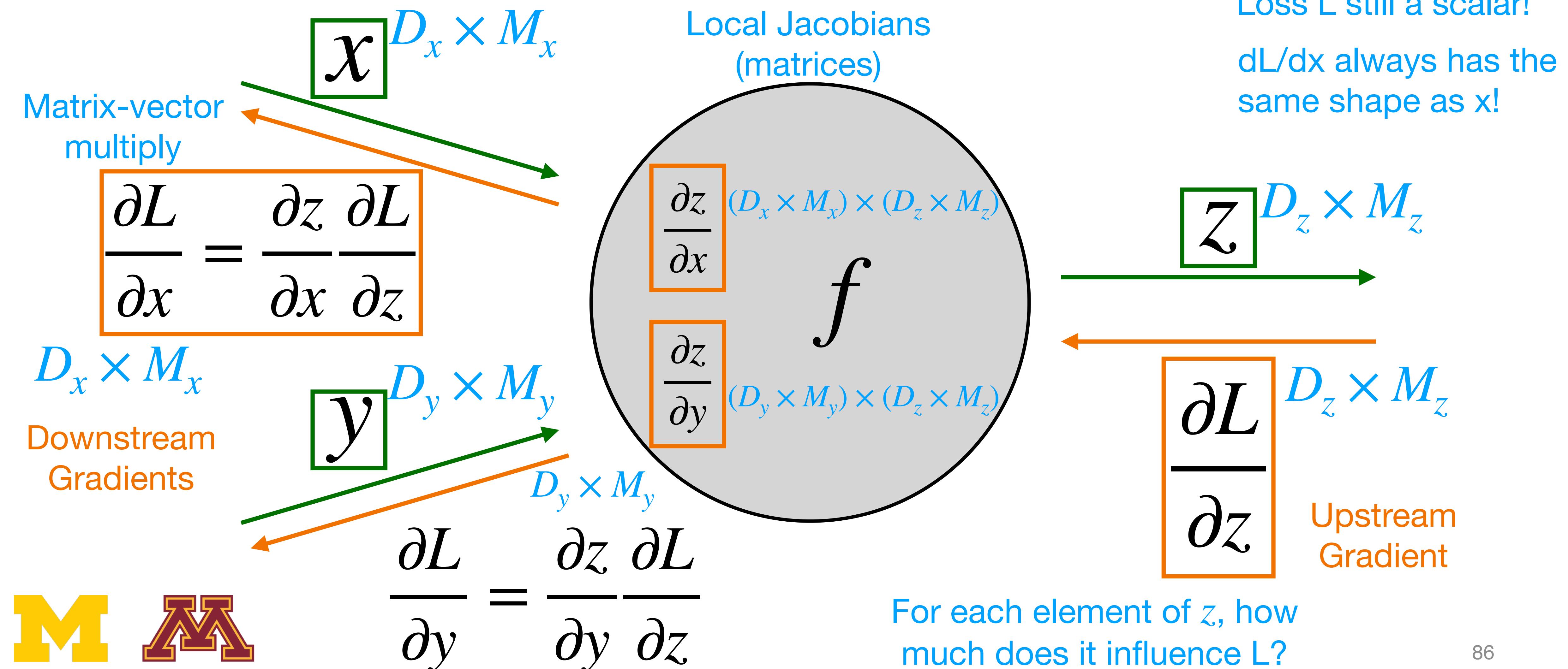
For each element of z , how much does it influence L ?

Backprop with Matrices (or Tensors)



For each element of z , how much does it influence L ?

Backprop with Matrices (or Tensors)



Example: Matrix Multiplication

$$\begin{array}{l} x: [N \times D] \\ \begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \end{array} \quad \begin{array}{l} w: [D \times M] \\ \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix} \end{array}$$



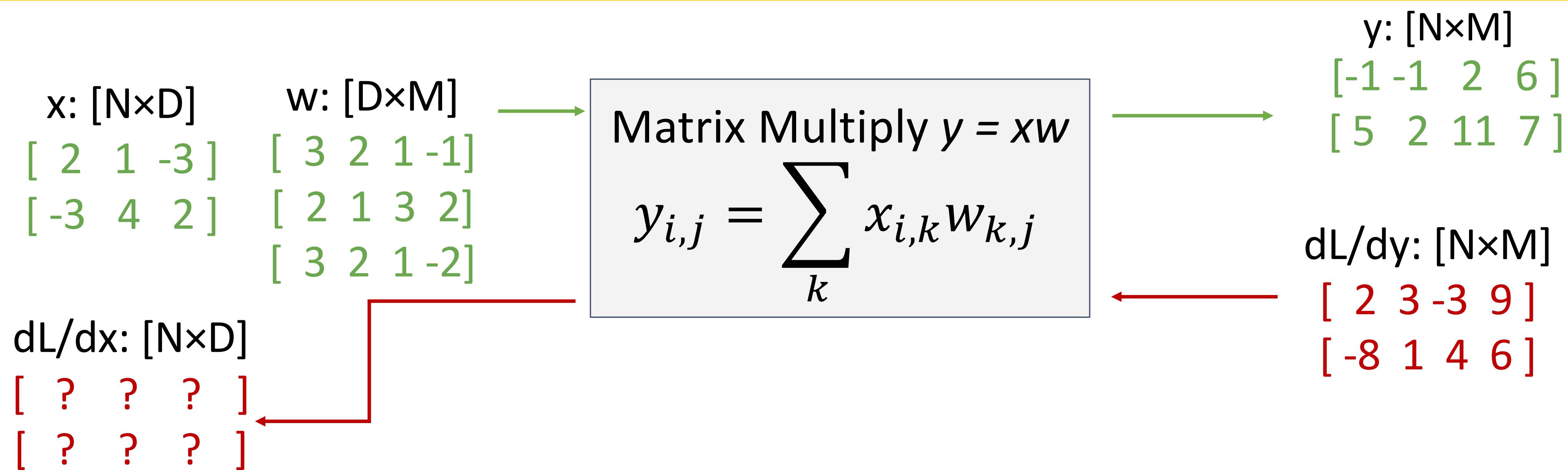
Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

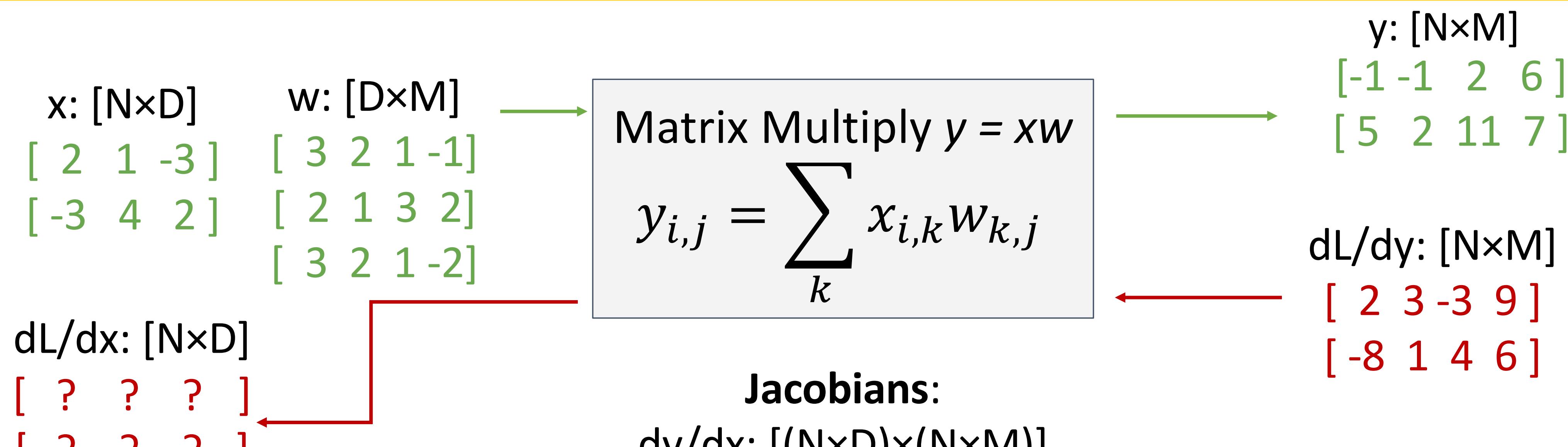


$$\begin{array}{l} y: [N \times M] \\ \begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix} \end{array}$$

Example: Matrix Multiplication



Example: Matrix Multiplication

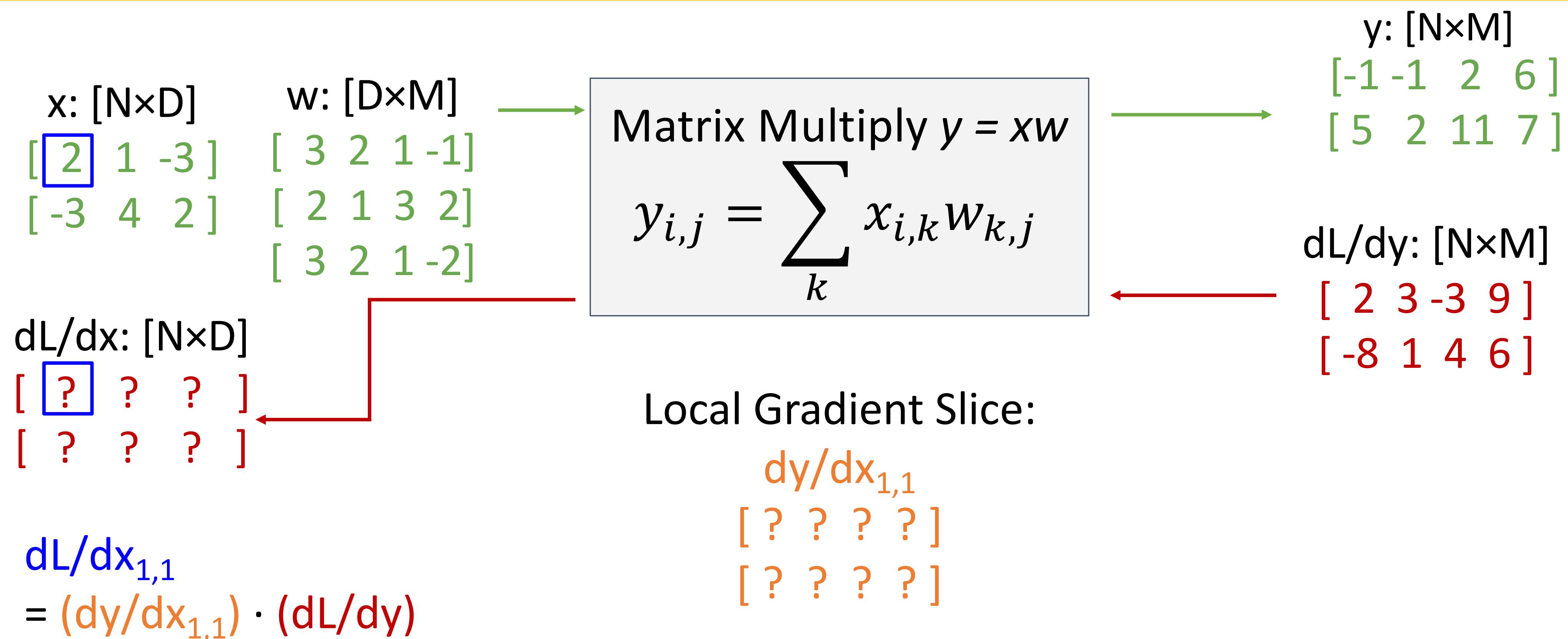


For a neural net we may have

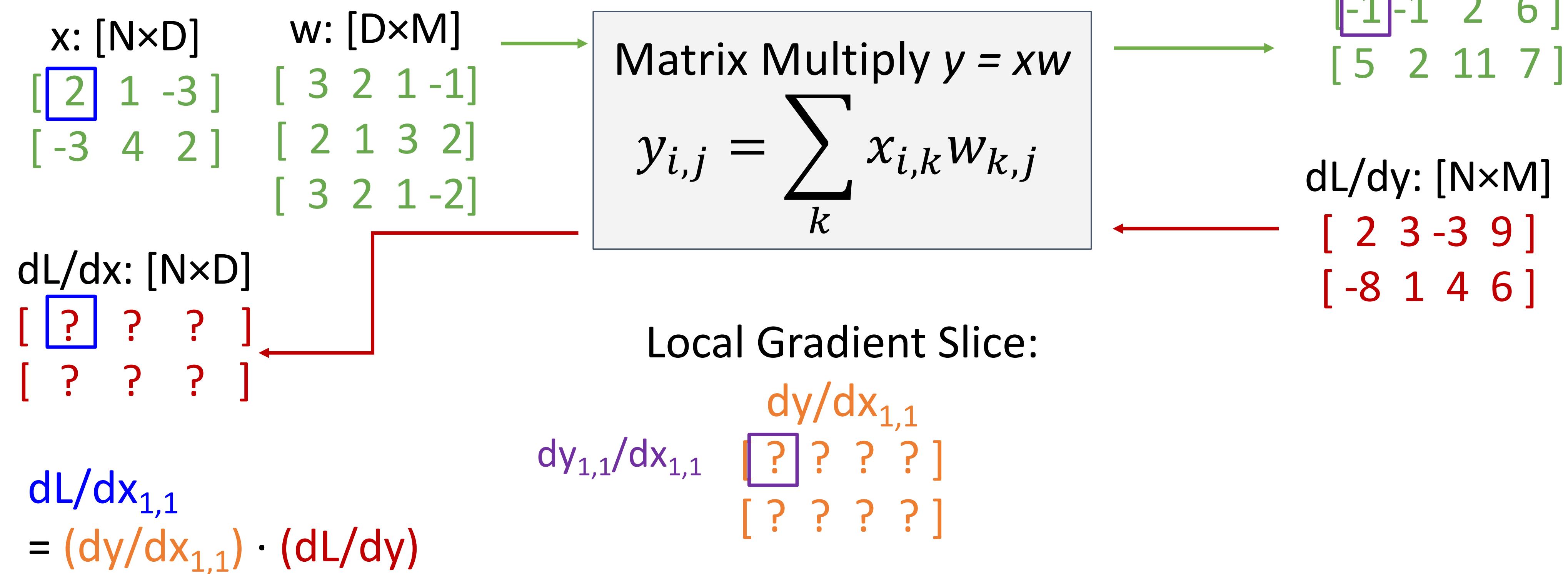
$$N=64, D=M=4096$$

Each Jacobian takes 256 GB of memory! Must work with them implicitly!

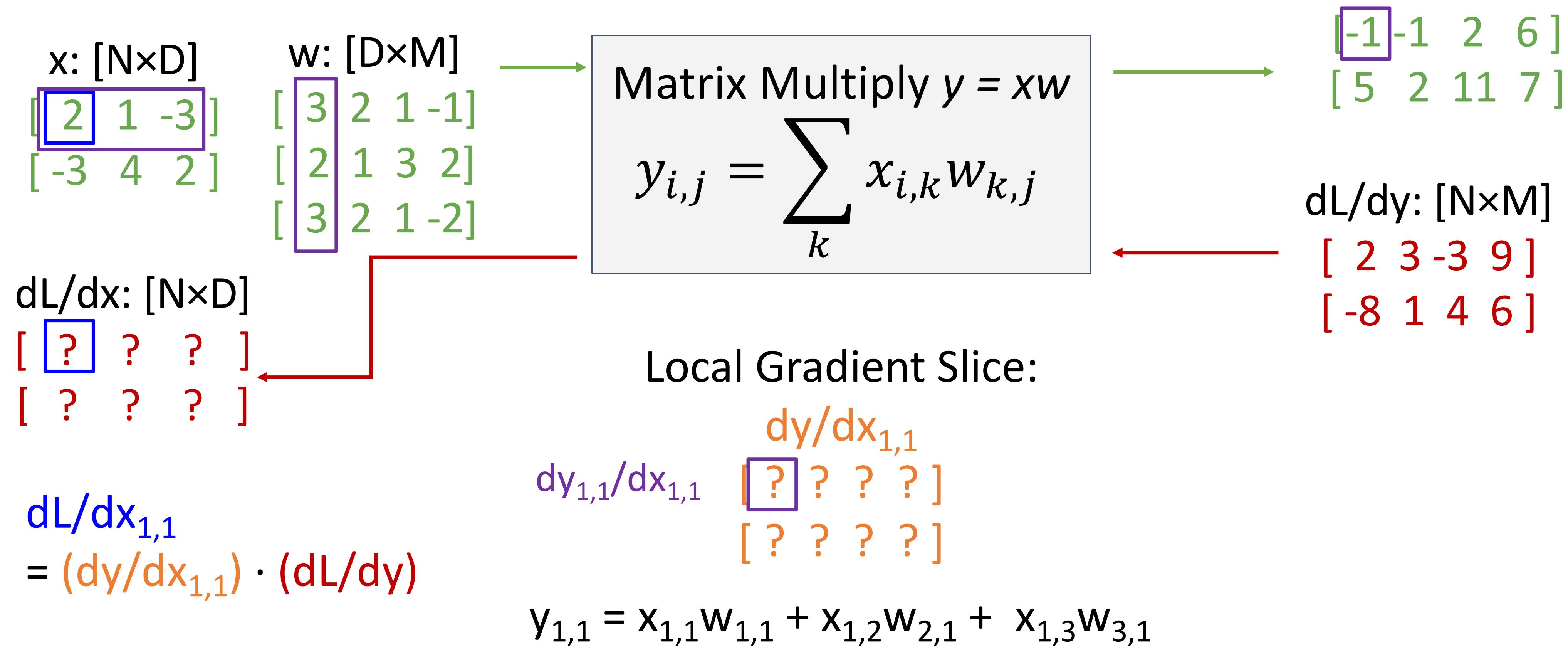
Example: Matrix Multiplication



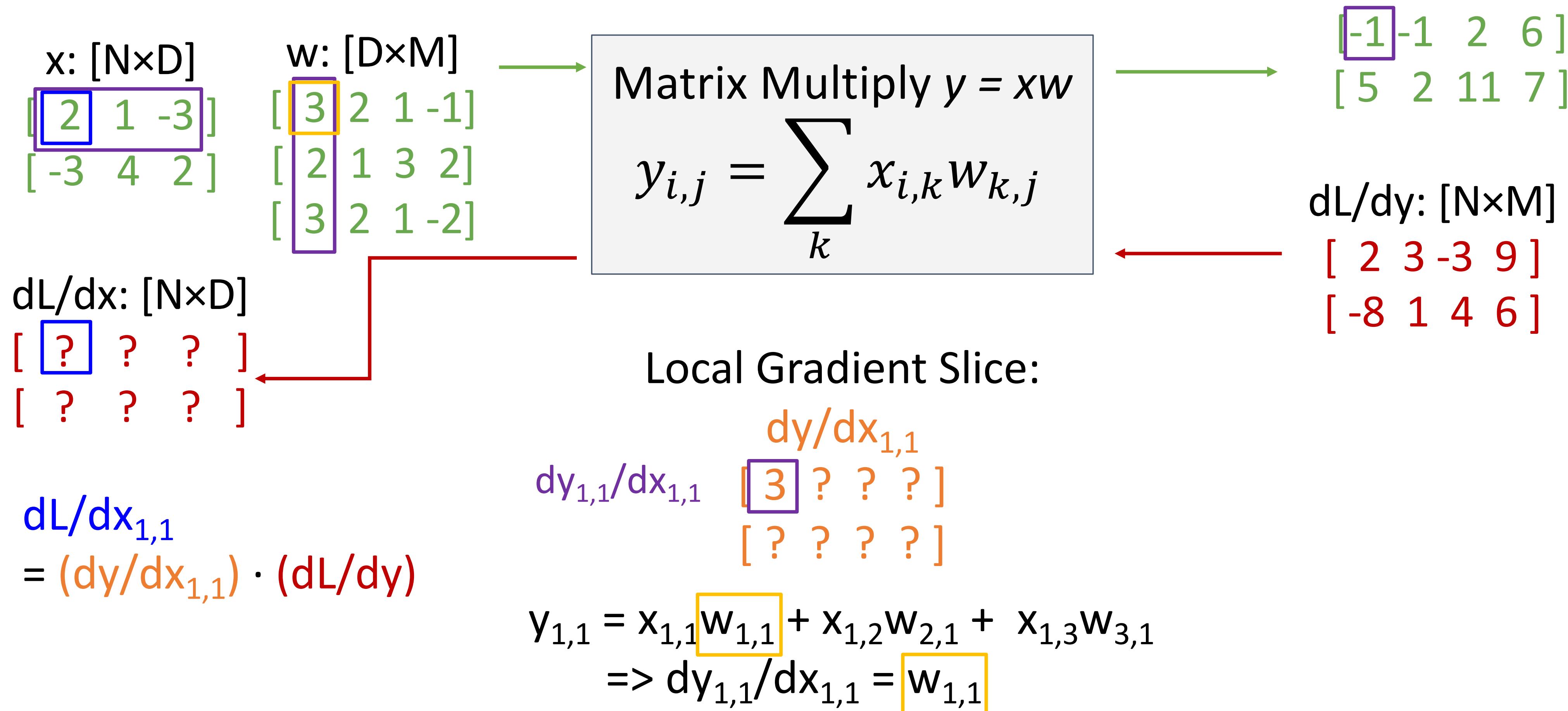
Example: Matrix Multiplication



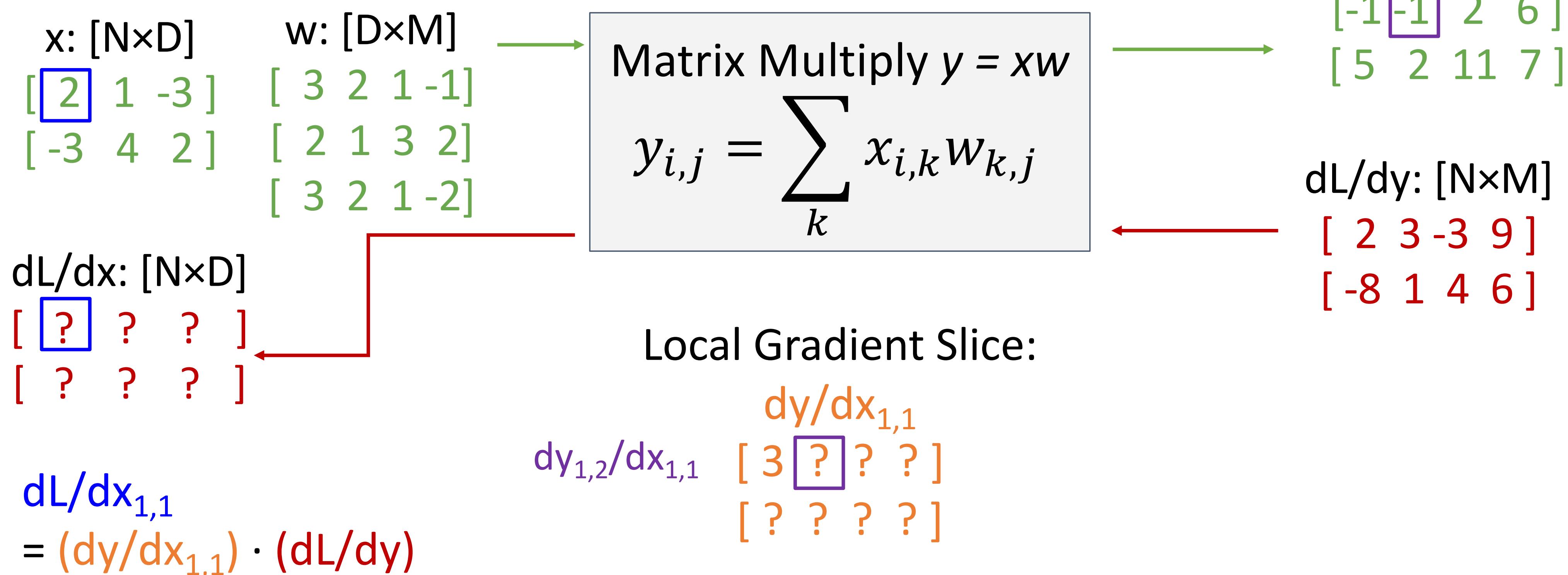
Example: Matrix Multiplication



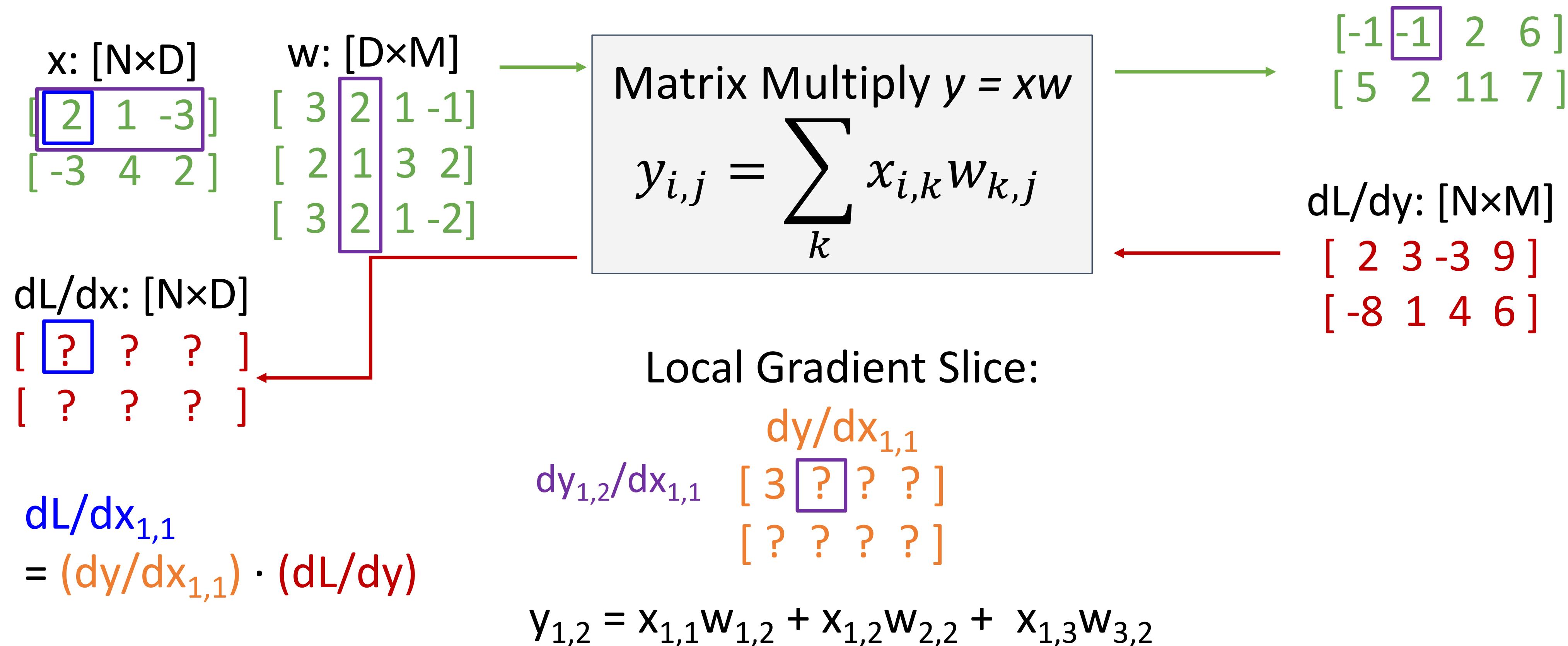
Example: Matrix Multiplication



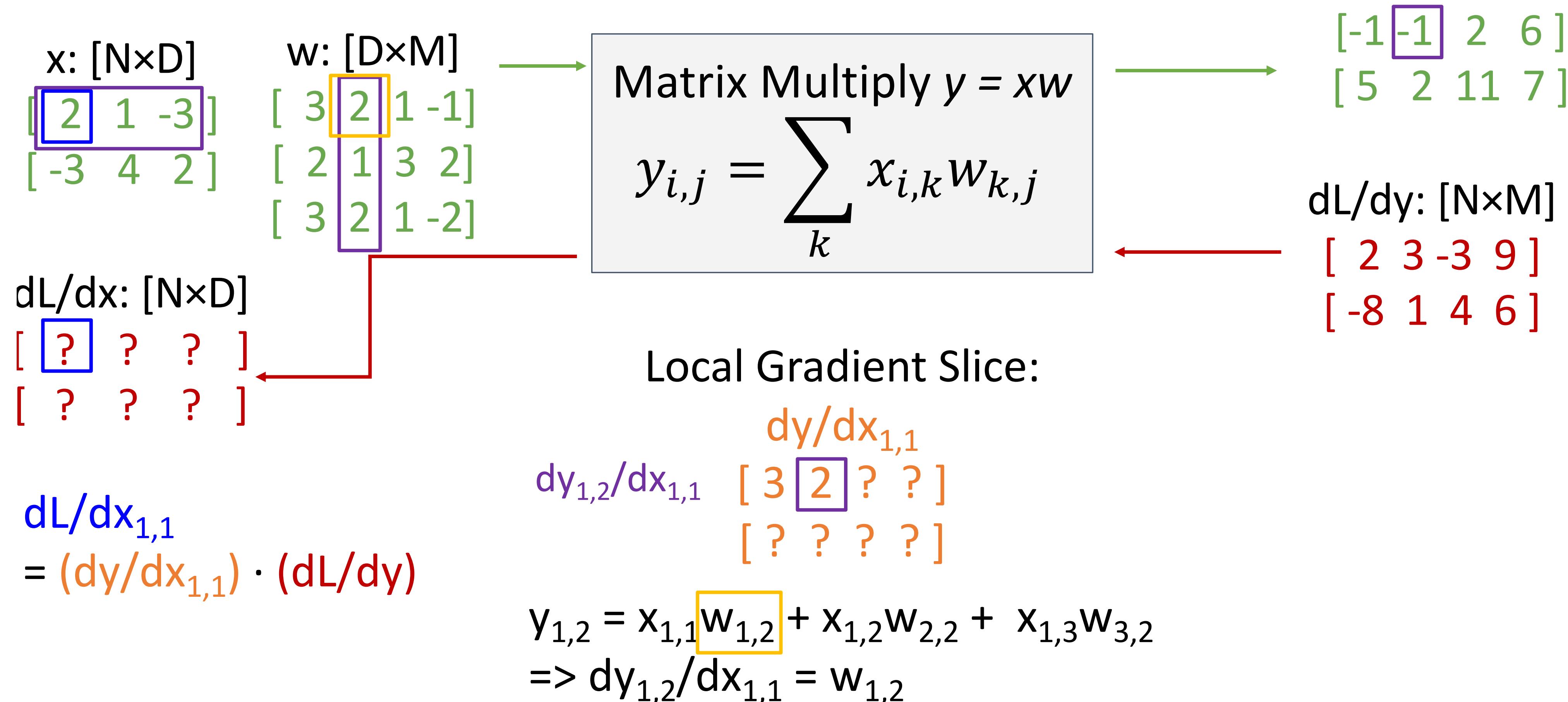
Example: Matrix Multiplication



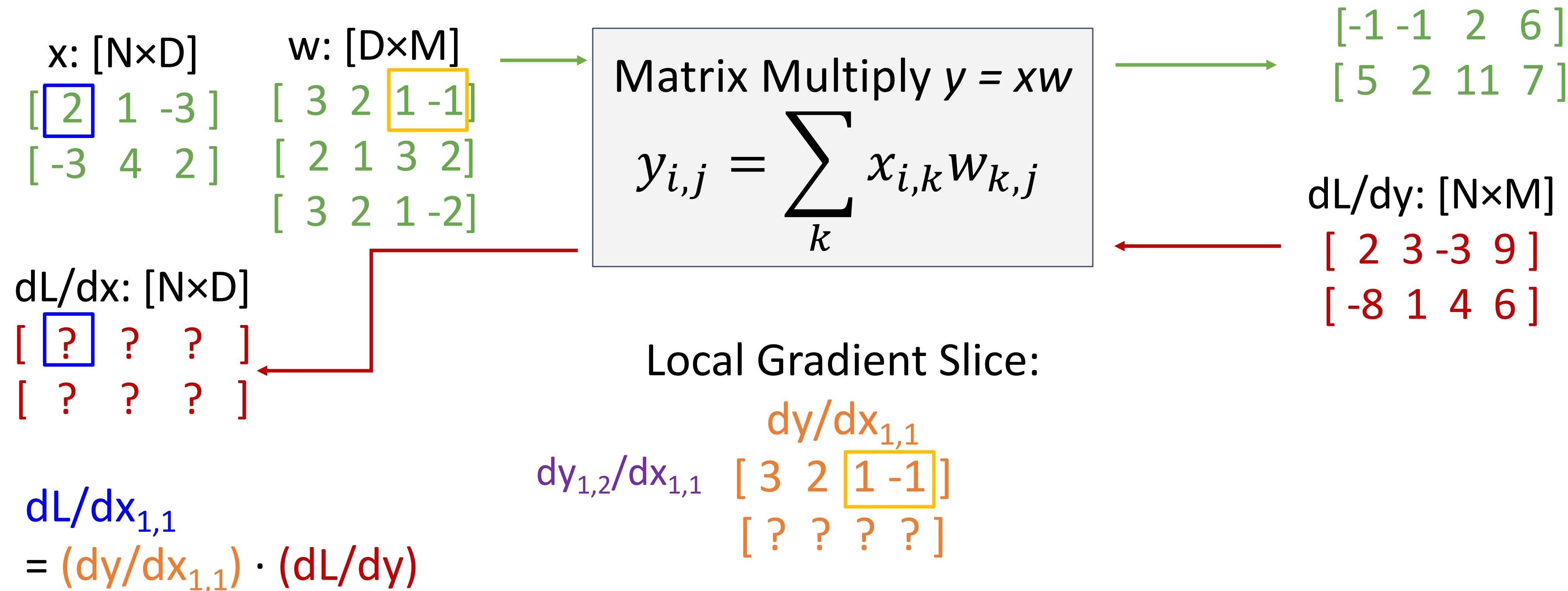
Example: Matrix Multiplication



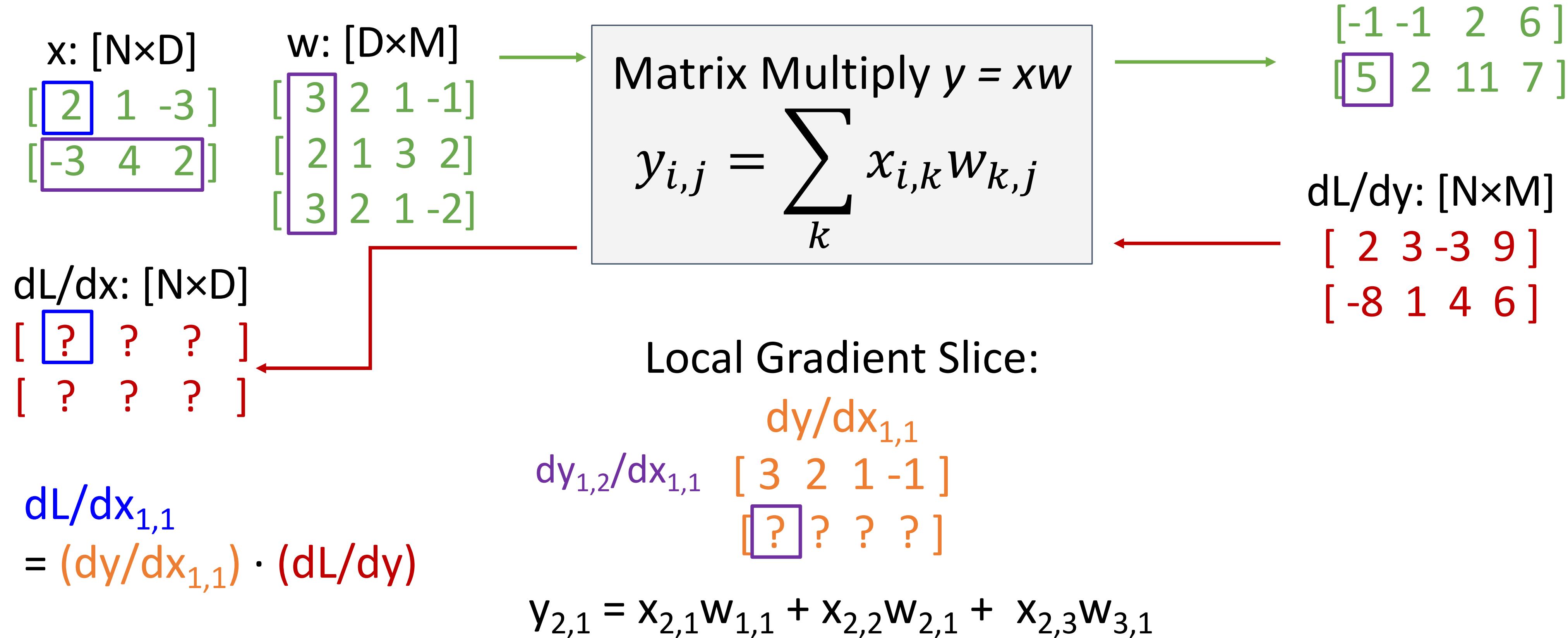
Example: Matrix Multiplication



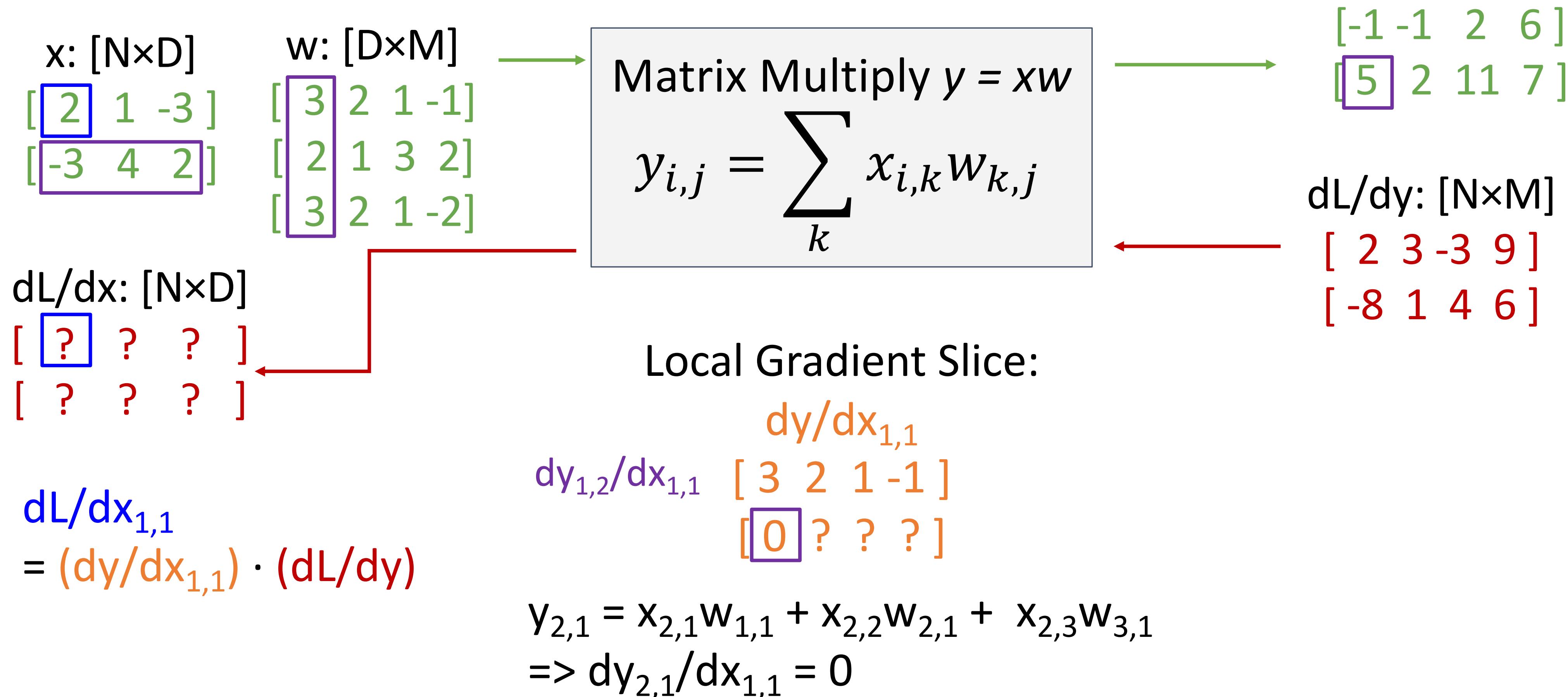
Example: Matrix Multiplication



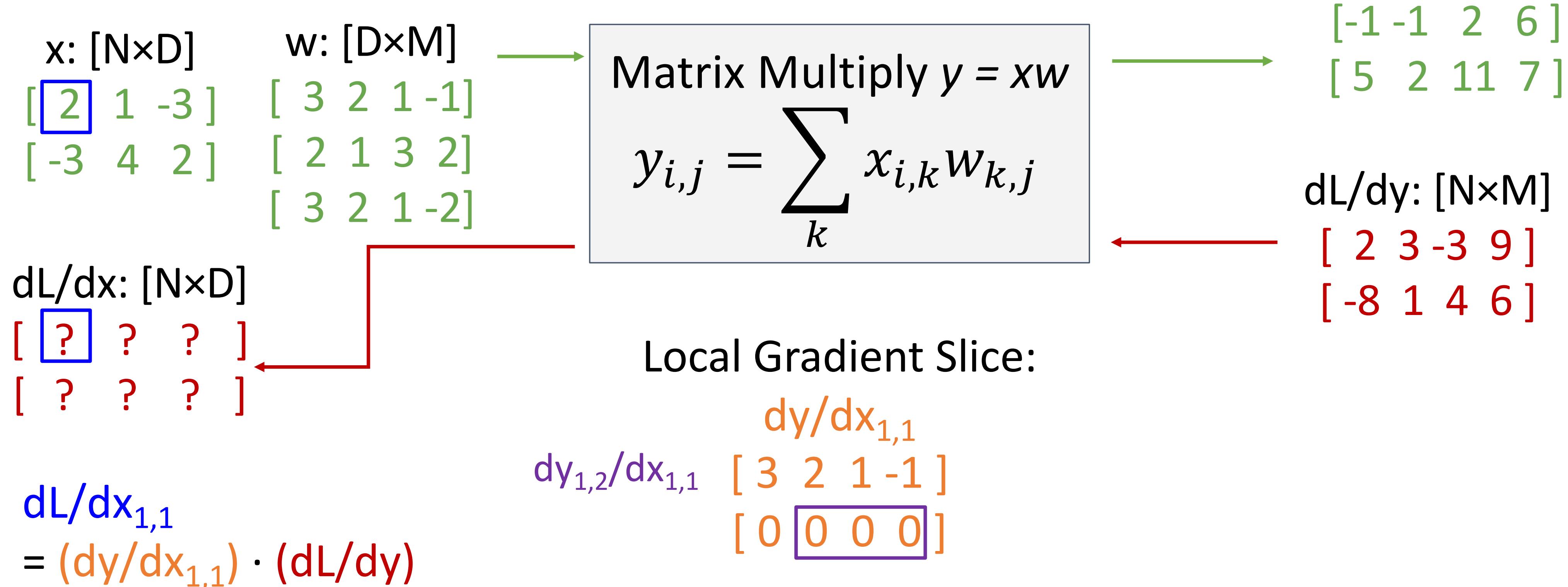
Example: Matrix Multiplication



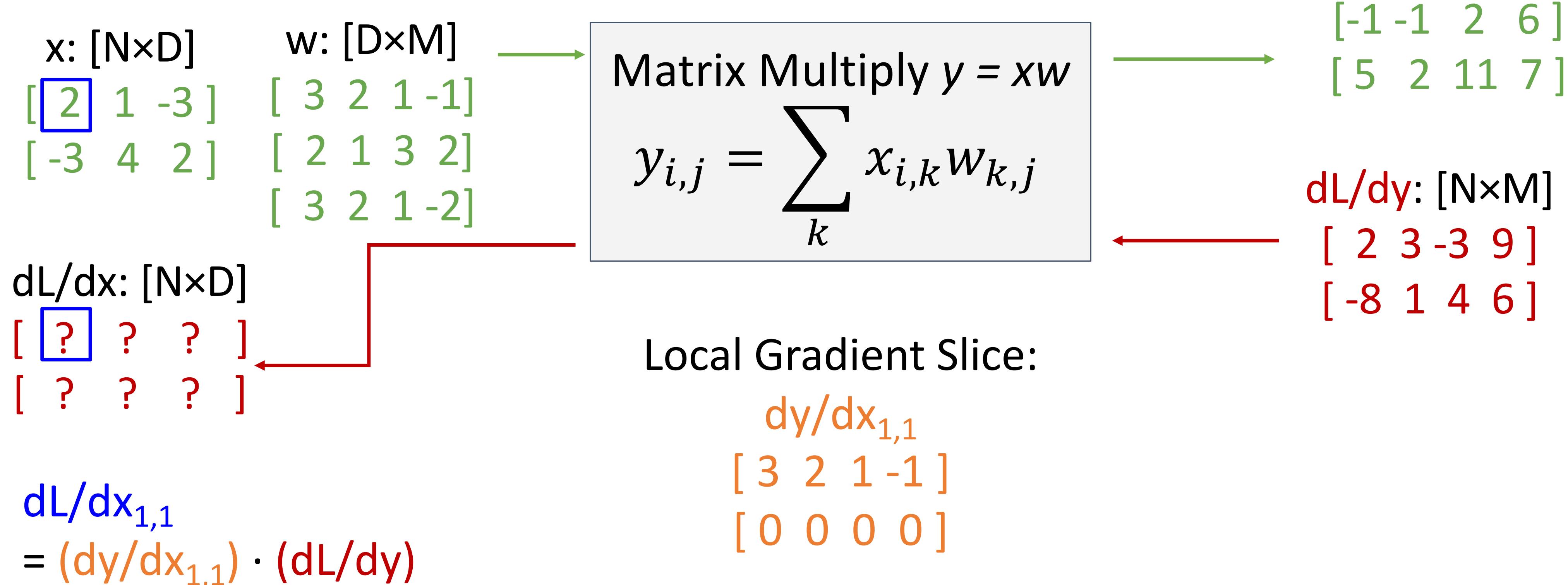
Example: Matrix Multiplication



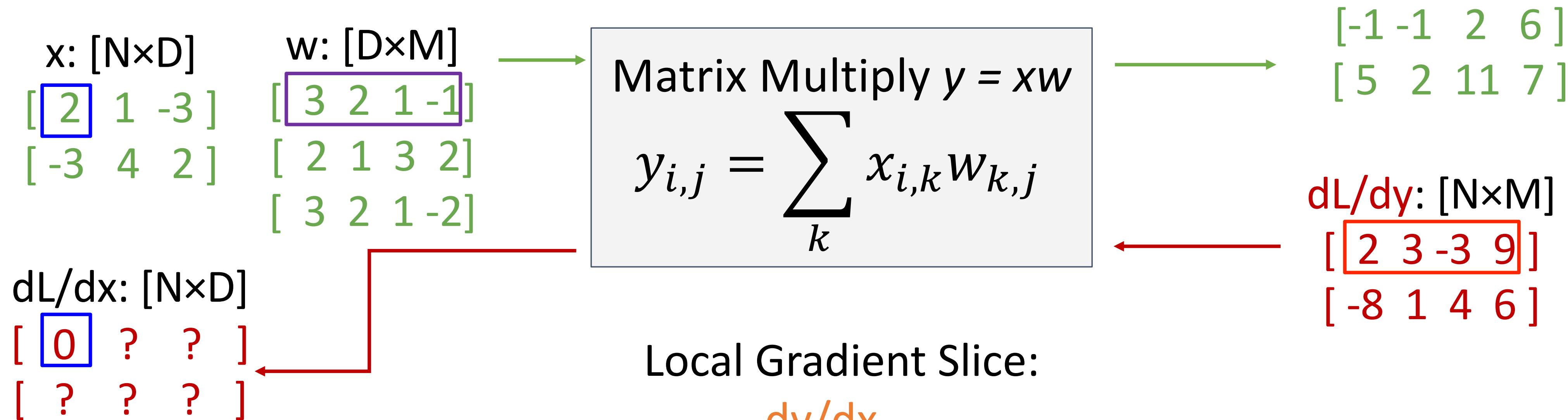
Example: Matrix Multiplication



Example: Matrix Multiplication



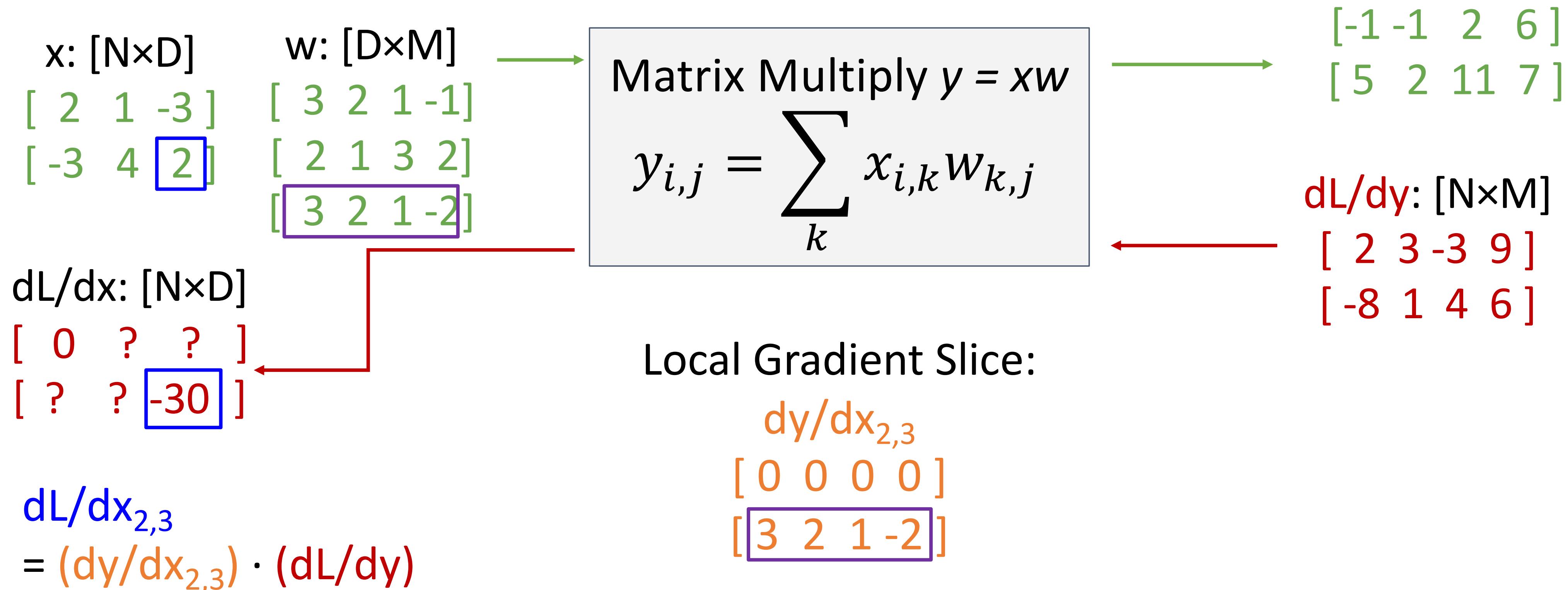
Example: Matrix Multiplication



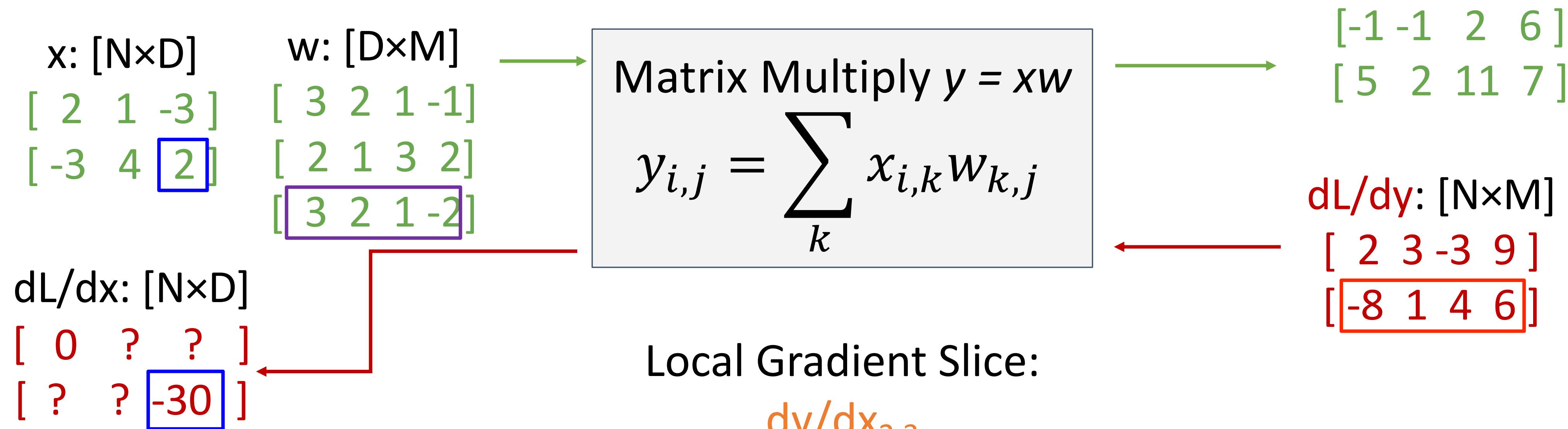
$$\begin{aligned}
 dL/dx_{1,1} &= (dy/dx_{1,1}) \cdot (dL/dy) \\
 &= (w_{1,:}) \cdot (dL/dy_{1,:}) \\
 &= 3*2 + 2*3 + 1*(-3) + (-1)*9 = 0
 \end{aligned}$$



Example: Matrix Multiplication



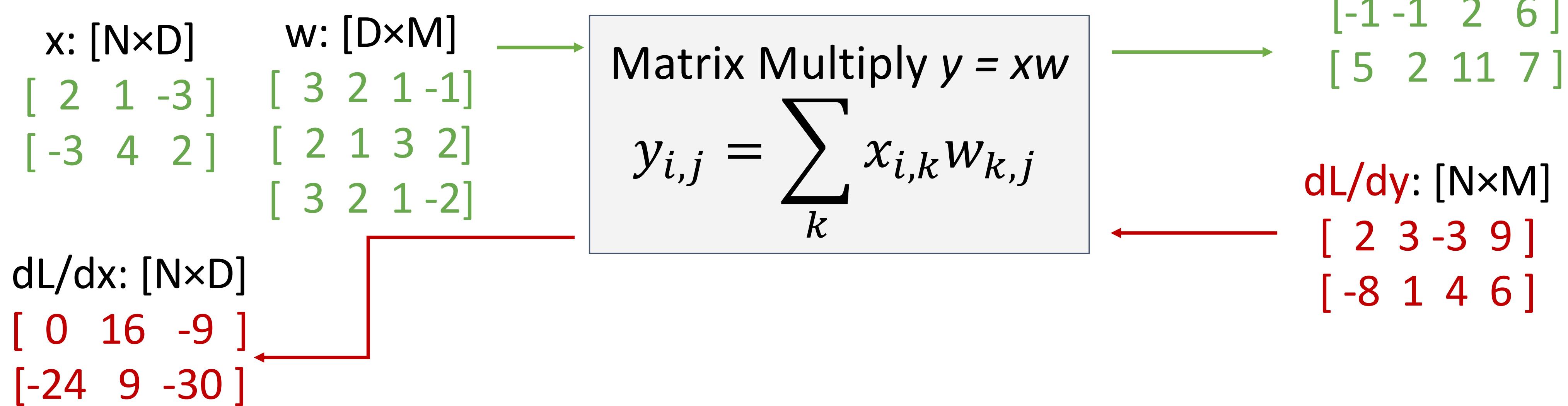
Example: Matrix Multiplication



$$\begin{aligned}
 dL/dx_{2,3} &= (dy/dx_{2,3}) \cdot (dL/dy) \\
 &= (w_{3,:}) \cdot (dL/dy_{2,:}) \\
 &= 3*(-8) + 2*1 + 1*4 + (-2)*6 = -30
 \end{aligned}$$

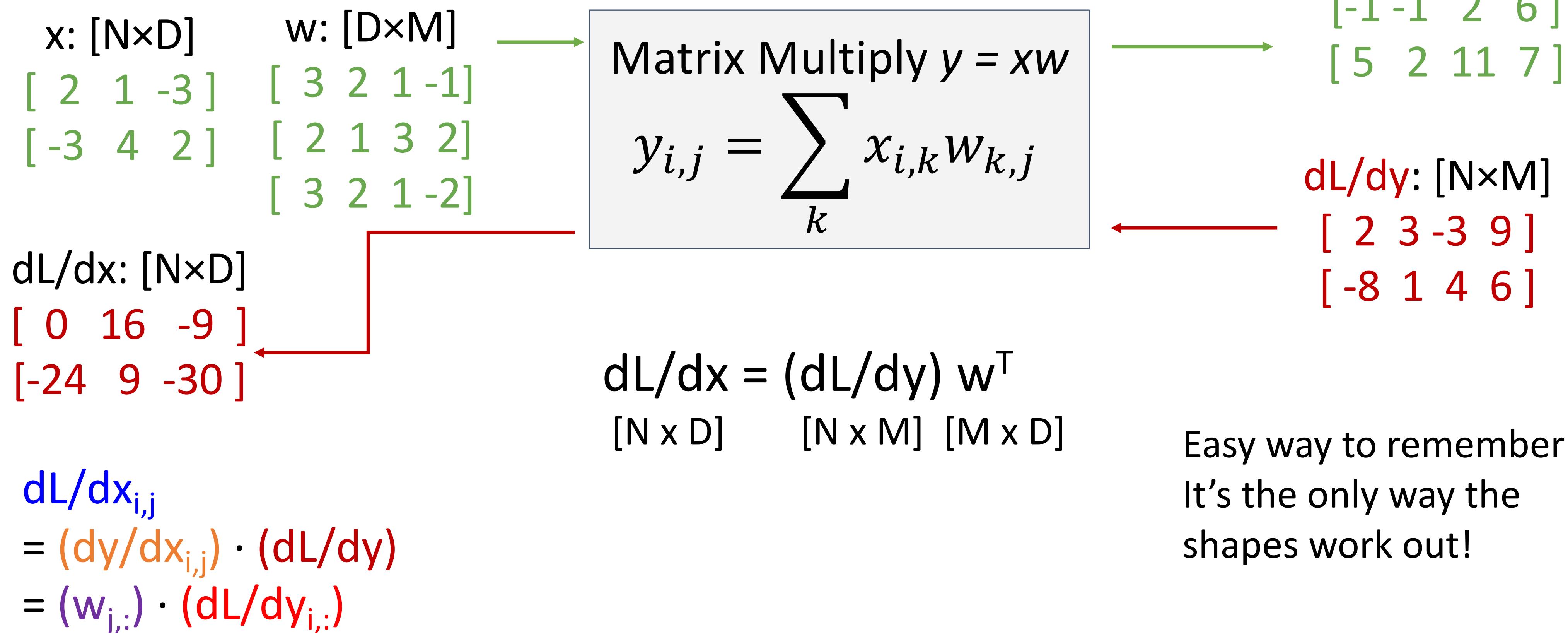


Example: Matrix Multiplication



$$\begin{aligned}
 dL/dx_{i,j} &= (dy/dx_{i,j}) \cdot (dL/dy) \\
 &= (w_{j,:}) \cdot (dL/dy_{i,:})
 \end{aligned}$$

Example: Matrix Multiplication



Example: Matrix Multiplication

$$\begin{array}{l} x: [N \times D] \\ \begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix} \end{array} \quad \begin{array}{l} w: [D \times M] \\ \begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix} \end{array}$$

Matrix Multiply $y = xw$

$$y_{i,j} = \sum_k x_{i,k} w_{k,j}$$

$$\begin{bmatrix} -1 & -1 & 2 & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

$$\begin{array}{l} dL/dx: [N \times D] \\ \begin{bmatrix} 0 & 16 & -9 \\ -24 & 9 & -30 \end{bmatrix} \end{array}$$

$$dL/dx = (dL/dy) w^T$$

$$[N \times D] \quad [N \times M] \quad [M \times D]$$

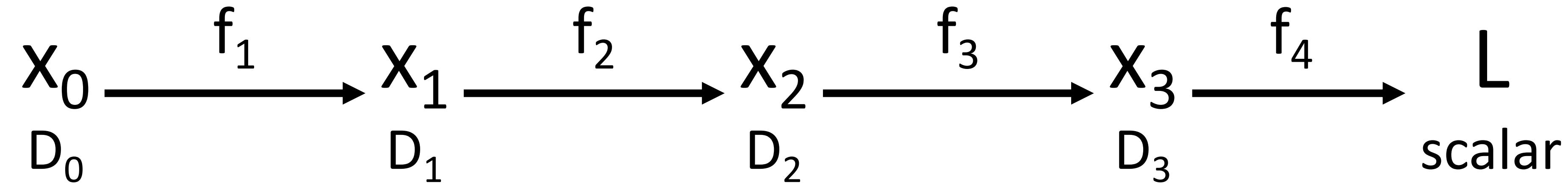
$$dL/dw = x^T (dL/dy)$$

$$[D \times M] \quad [D \times N] \quad [N \times M]$$

$$\begin{array}{l} dL/dy: [N \times M] \\ \begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix} \end{array}$$

Easy way to remember:
It's the only way the
shapes work out!

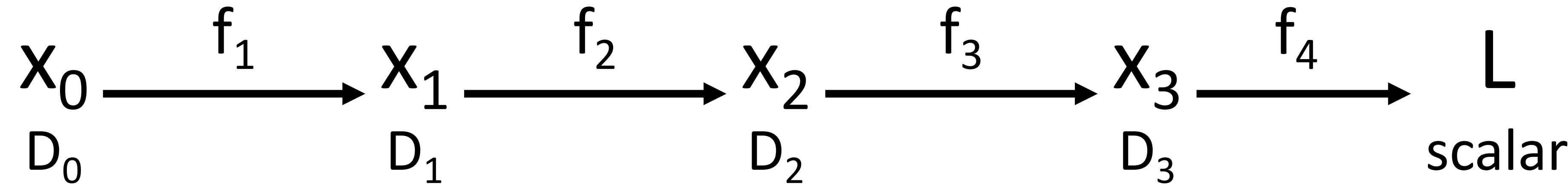
Backpropagation: Another View



Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

Backpropagation: Another View



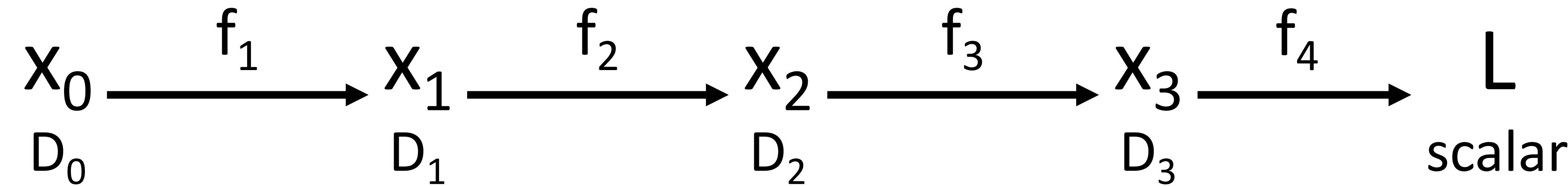
Matrix multiplication is **associative**: we can compute products in any order

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$

Reverse-Mode Automatic Differentiation



Matrix multiplication is **associative**: we can compute products in any order

Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

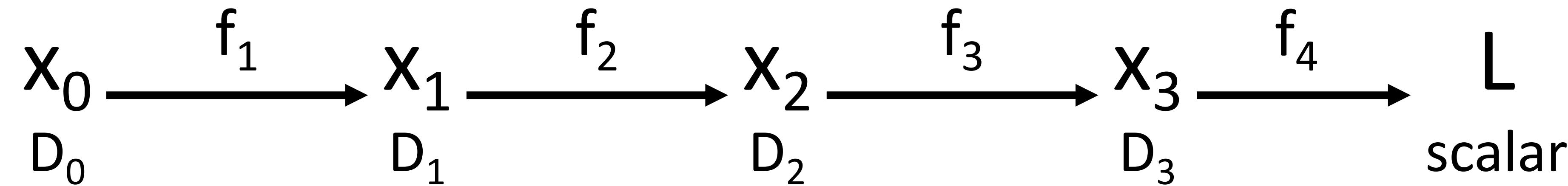
←

Chain rule

$$\frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

$$[D_0 \times D_1] \ [D_1 \times D_2] \ [D_2 \times D_3] \ [D_3]$$

Reverse-Mode Automatic Differentiation



Matrix multiplication is **associative**: we can compute products in any order

Computing products right-to-left avoids matrix-matrix products; only needs matrix-vector

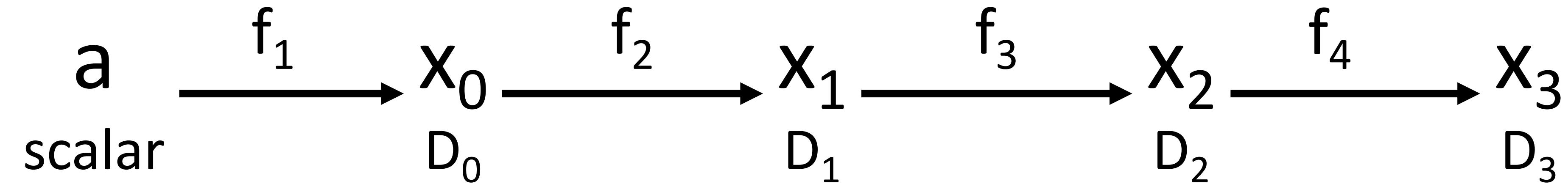
$$\text{Chain rule} \quad \frac{\partial L}{\partial x_0} = \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right) \left(\frac{\partial L}{\partial x_3} \right)$$

Compute grad of scalar output
w/respect to all vector inputs

$[D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3] [D_3]$

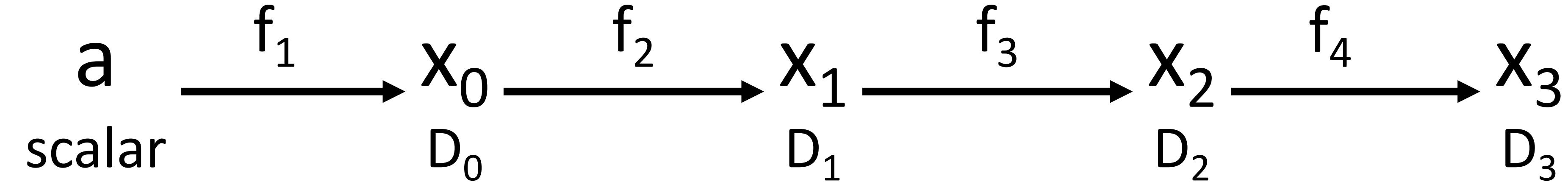
What if we want
grads of scalar
input w/respect
to vector
outputs?

Forward-Mode Automatic Differentiation



Chain rule
$$\frac{\partial x_3}{\partial a} = \left(\frac{\partial x_0}{\partial a} \right) \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right)$$
$$[D_0] [D_0 \times D_1] [D_1 \times D_2] [D_2 \times D_3]$$

Forward-Mode Automatic Differentiation



Computing products left-to-right avoids matrix-matrix products; only needs matrix-vector

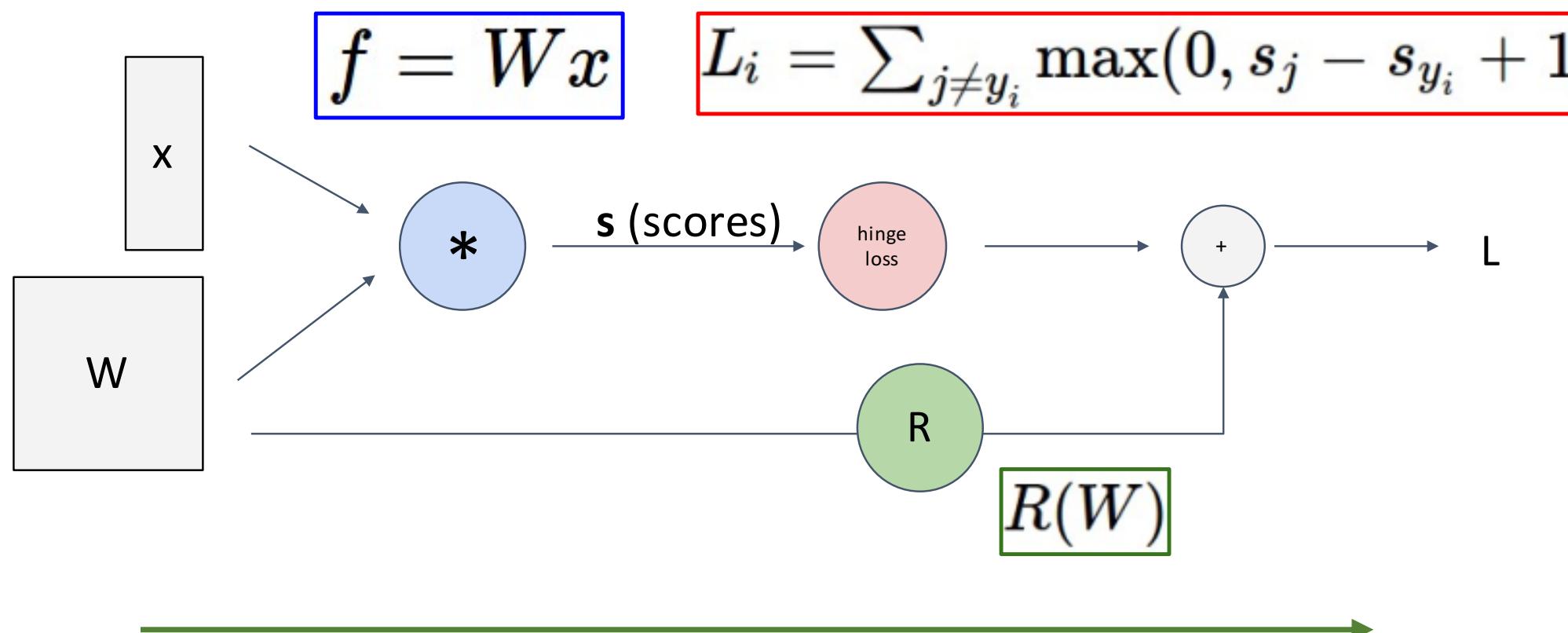
Chain rule

$$\frac{\partial x_3}{\partial a} = \overbrace{\left(\frac{\partial x_0}{\partial a} \right) \left(\frac{\partial x_1}{\partial x_0} \right) \left(\frac{\partial x_2}{\partial x_1} \right) \left(\frac{\partial x_3}{\partial x_2} \right)}^{\longrightarrow}$$

$[D_0] \quad [D_0 \times D_1] \quad [D_1 \times D_2] \quad [D_2 \times D_3]$

Summary

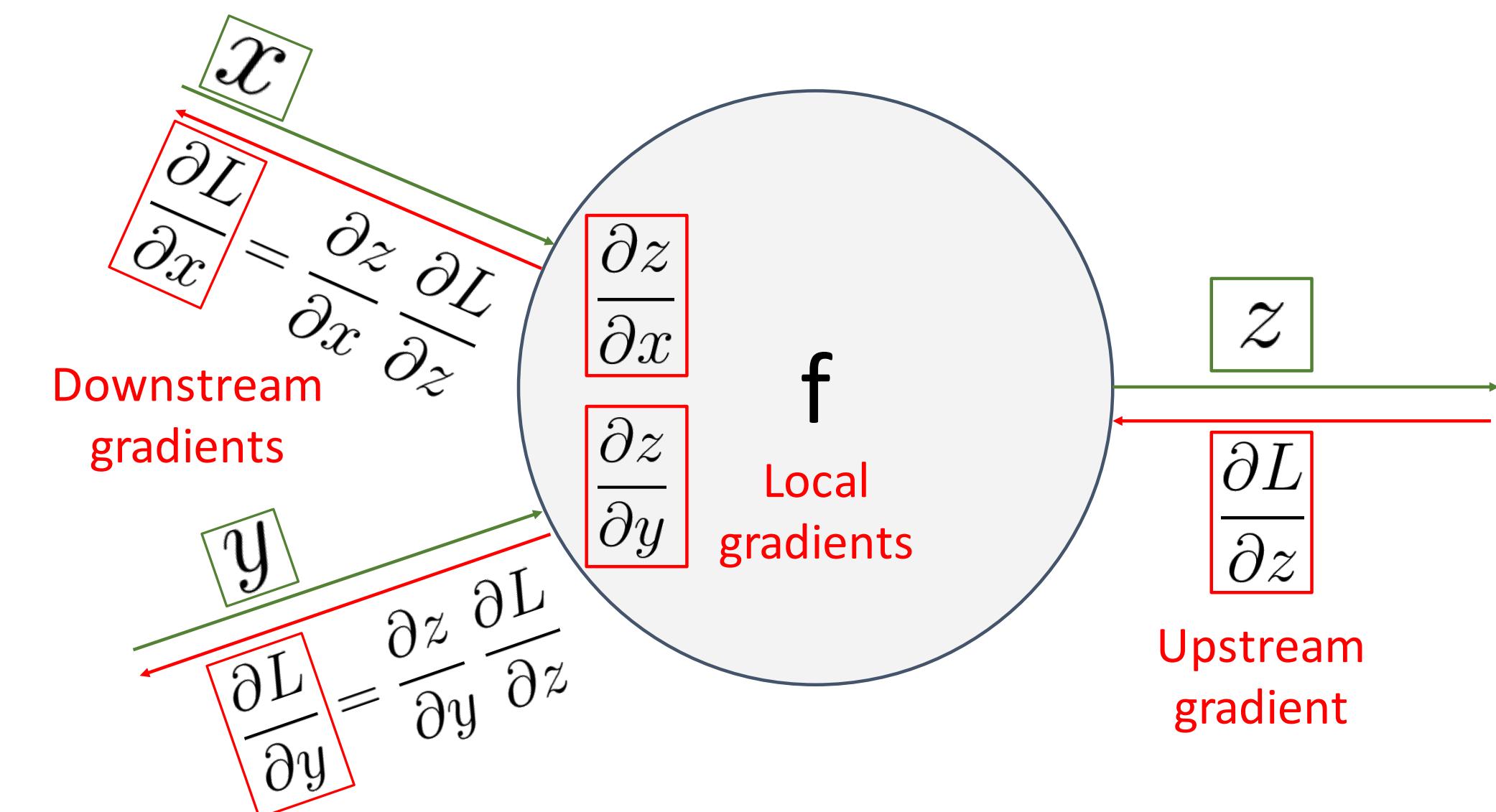
Represent complex expressions as **computational graphs**



Forward pass computes outputs

Backward pass computes gradients

During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**



Summary

Backprop can be implemented with “flat” code where the backward pass looks like forward pass reversed (Use this for A2!)

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

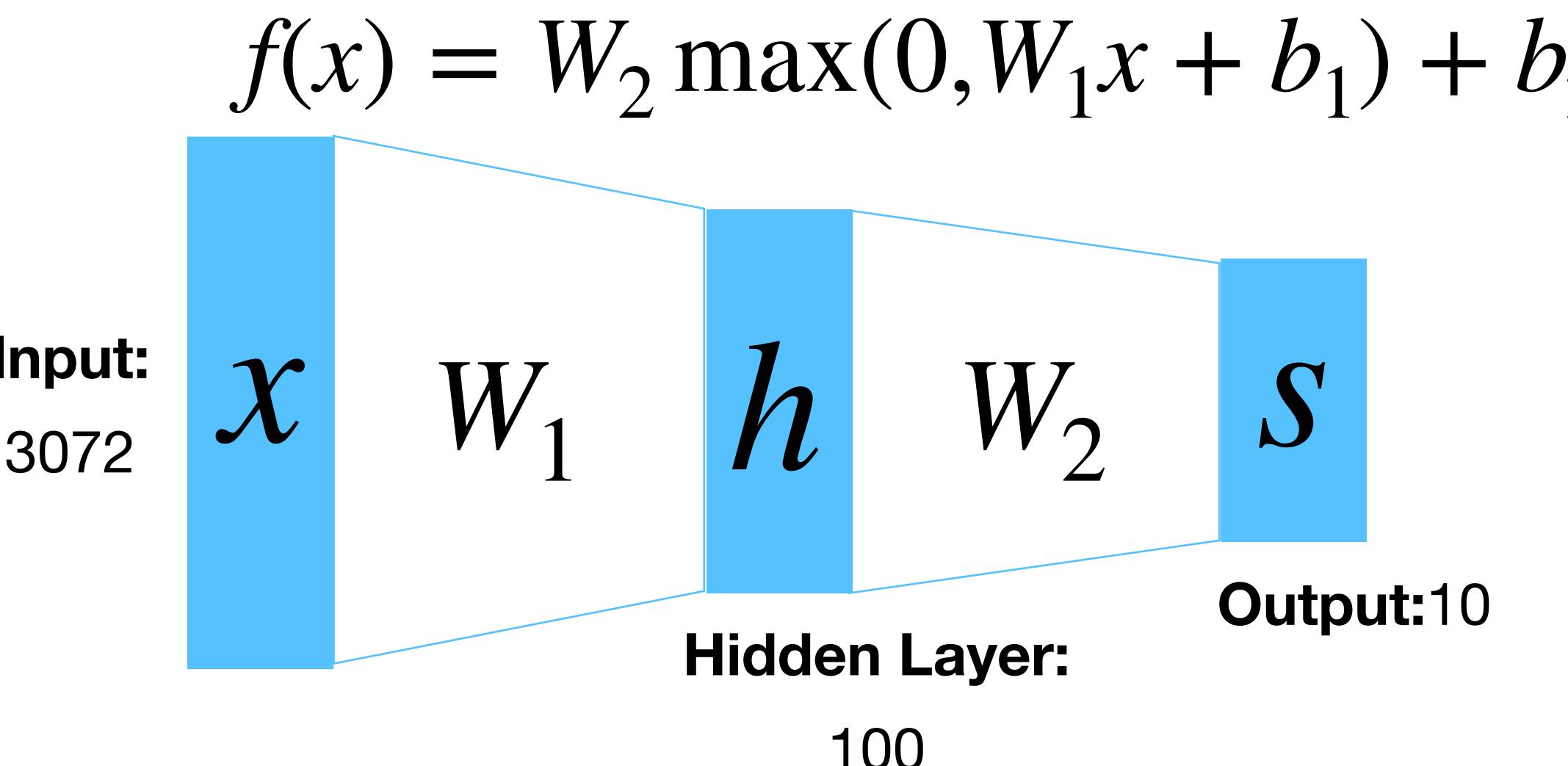
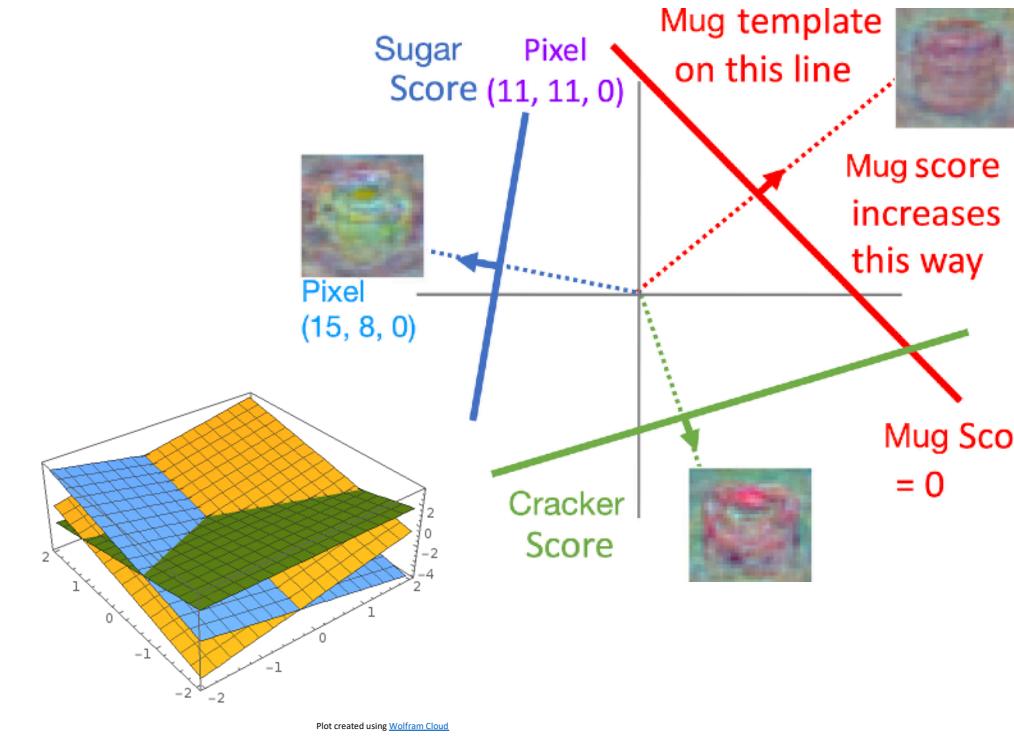
Backprop can be implemented with a modular API, as a set of paired forward/backward functions

```
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        z = x * y
        return z

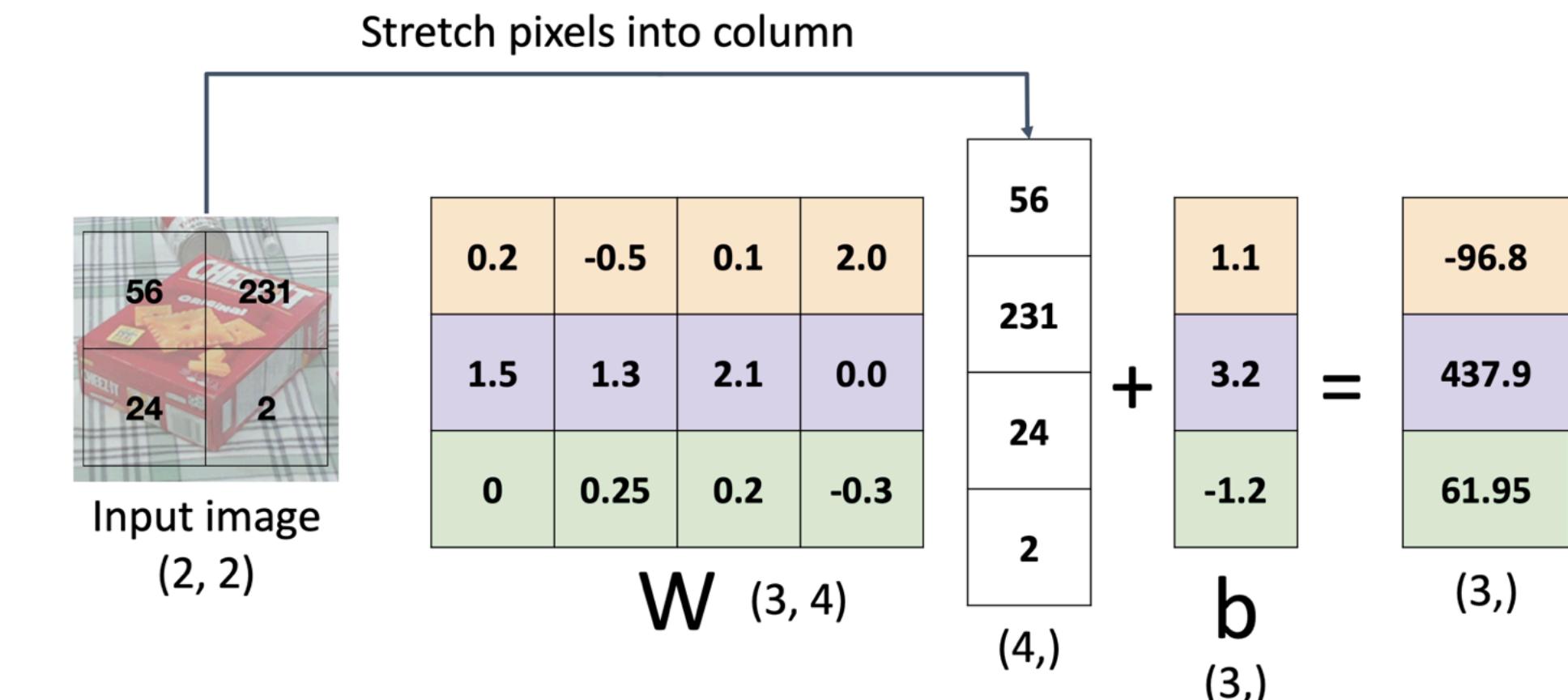
    @staticmethod
    def backward(ctx, grad_z):
        x, y = ctx.saved_tensors
        grad_x = y * grad_z    # dz/dx * dL/dz
        grad_y = x * grad_z    # dz/dy * dL/dz
        return grad_x, grad_y
```



Summary



Problem: So far our classifiers don't respect the spatial structure of images!



Next time: Convolutional Neural Networks



DR

DeepRob

Lecture 6
Backpropagation
University of Michigan and University of Minnesota

$$\frac{\partial L}{\partial W_{\ell_1}}$$

$$\frac{\partial L}{\partial W_{\ell_2}}$$

$$\frac{\partial L}{\partial W_{\ell_3}}$$

$$\frac{\partial L}{\partial W_{\ell_4}}$$

$$\frac{\partial L}{\partial W_{\ell_5}}$$

$$\frac{\partial L}{\partial \text{Out}}$$

